UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

DEEP REINFORCEMENT LEARNING IA PARA STARCRAFT 2

Director: Javier Béjar Alonso

Carlos Roldán Montaner

Junio 2018

Resumen

La inteligencia artificial y los videojuegos son campos que van de la mano, ya que el segundo supone un campo de entrenamiento excepcional para el primero. Tras conquistar numerosos juegos de diversos tipos, el nuevo gran reto de la IA es *Starcraft 2*. Este proyecto intenta desarrollar agentes de aprendizaje por refuerzo para escenarios concretos de *Starcraft 2* y sacar conclusiones sobre cuales son las dificultades más importantes y los enfoques más adecuados para afrontar el reto que supone.

Resum

La intel · ligència artificial i els videojocs van de la mà, ja que el segon suposa un camp d'entrenament excepcional pel primer. Després de conquerir nombrosos jocs de diferents tipus, el nou gran repte de la IA és *Starcraft 2*. Aquest projecte té com a objectiu desenvolupar agents d'aprenentatge per reforç per a escenaris específics d'*Starcraft 2* i treure conclusions sobre quines són les dificultats més importants i les perspectives més adequades per enfrontar el repte que suposa.

Abstract

Artificial intelligence and videogames go hand in hand, because the latter are a great training field for the former. After conquerir numerous games of different kinds, the new big challenge for AI is *Starcraft 2*. This project's goal is to develop reinforcement learning agents for specific scenarios of *Starcraft 2* and to draw some conclusions about which are the main difficulties and what are the best approaches to face the challenge that it presents.

Agradecimientos

Me gustaría dar las gracias a:

Mi tutor y profesor de Inteligencia Artificial y CAIM, Javier Béjar, por guiarme y asesorarme a lo largo de este proyecto, responder a infinidad de e-mails de forma incansable y sobretodo por descubrirme la inteligencia artificial y contagiarme su pasión por la misma.

Mi familia, por haber estado a mi lado, animado a estudiar lo que me gustaba y haberme apoyado a lo largo de todos mis años de estudiante.

Y a mis amigos, por estar siempre ahí, regalándome risas y buenos momentos. No puedo nombraros a todos porque no acabaría pero: a Juanmi, por estar ahí desde el primer minuto literalmente y todo lo que me ha ayudado; a Mario, porque ha sido como un hermano mayor (o pequeño a veces); a Carlota, por preocuparse siempre y ahorrarme tantas horas de transporte público; a David, por mantener la población de ardillas bajo control; a Jordi, porque sin sus juegos de mesa no seríamos nada y por acercarme siempre a casa; a Dídac, por hacer faena de más para que yo hiciese de menos y por la pizza de kebab; a Ana, por recomendarme tan buena música, por hacerme reír y por tener siempre una palabra de ánimo; y a todos los demás, que ya sabéis quiénes sois, por estos últimos cuatro años que han sido la mejor etapa de mi vida.

Gracias.

Índice general

Capítulo 1

Contexto y alcance del proyecto

1.1. Contexto y formulación del problema

La inteligencia artificial es uno de los pilares principales de la computación y cobra cada vez más importancia. Elementos que hace no tantos años habrían sido tildados de ciencia ficción, son hoy en día parte de nuestras vidas y los avances futuros son muy prometedores.

Uno de los campos que más ha contribuido al desarrollo de la IA es sin duda el de los juegos, y en especial el de los videojuegos. Estos últimos se presentan desde sus orígenes como el dominio ideal, el mejor campo de pruebas para el estudio de la IA por diversos motivos.

Uno de ellos es que los videojuegos ofrecen una de las formas de interacción persona-computador más ricas y complejas. Esta riqueza viene definida por el gran número de opciones que el jugador tiene disponible en cada momento, así como por las diversas maneras de interactuar con el entorno. El número de posibilidades está normalmente relacionado con la complejidad del juego de manera directa: a mayor complejidad, mayor número de opciones disponibles para el jugador en cada momento.

La popularidad de los videojuegos es otro factor influyente. A diferencia de hace unas décadas, cuando la disponibilidad de los videojuegos era limi-

tada y la opción más común eran las recreativas, hoy en día los videojuegos están en cualquier parte, ordenadores, consolas, tablets, móviles... Existe por tanto una enorme industria detrás de este sector, impulsándolo tecnológica y económicamente y con él, a la IA.

Además, el incremento en popularidad, cantidad y complejidad de los videojuegos se traduce en dos cosas: un aumento de la cantidad y variedad de contenidos, que empuja las fronteras de la IA; y una enorme cantidad de datos obtenidos mediante técnicas de telemetría que tras ser procesados mediante técnicas de big data y big data mining research, constituyen un valioso recurso para las IAs basadas en Machine Learning. Por ejemplo, en marzo de 2017, el proyecto **Open Dota** lanzó un archivo conteniendo más de un billón de partidas de *Dota 2* (Valve Corporation, 2013), jugadas entre marzo de 2011 y marzo de 2016.

Finalmente, un factor clave (tal vez el más importante) es la dificultad de los videojuegos. Estos presentan problemas complejos debido a que sus espacios de estados finitos, como las posibilidades que tiene un agente, suelen ser muy extensos, mientras que los espacios de soluciones suelen ser pequeños. Además, en ocasiones es extremadamente difícil o imposible estimar la calidad de un estado de juego concreto en un instante determinado.

A lo largo del tiempo, distintos juegos han constituido *benchmarks* para los algoritmos de IA. El ajedrez lo fue hasta que Deep Blue tuvo éxito derrotando al Gran Maestro Gary Kásparov. Así, nuevos juegos más complejos como *Lemmings*, *Super Mario Bros* o *Starcraft* fueron tomados como *benchmarks*. En mayo de 2017, AlphaGo derrotó al campeón mundial de Go, Ke Jie 3-0. En agosto de 2017, la compañía OpenAI de Elon Musk presentó una IA que derrotó al jugador profesional de *Dota 2* Danylo "*Dendi*" Ishutin ante una asombrada multitud.[1]

De esta manera, *Starcraft* (Blizzard Entertainment), considerado por muchos el juego más difícil de la historia (y en particular su segunda iteración, *Starcraft 2*, en el que se centra este proyecto), se ha convertido en el siguiente gran objetivo de la IA en cuanto a videojuegos.

Conseguir una IA que domine *Starcraft 2* es, dada su extremada dificultad, un objetivo poco factible actualmente. Los mejores bots actuales apenas

son capaces de llegar al nivel de un jugador principiante. El objetivo de este proyecto es experimentar con distintos métodos de aprendizaje por refuerzo sobre escenarios específicos del juego, utilizando distintas funciones de refuerzo para comprobar que nivel puede obtener la IA y hacer los ajustes necesarios para alcanzar un nivel tan óptimo cómo sea posible. Así, la IA sería competente en los escenarios concretos que tienen lugar en una partida típica, a pesar de no dominar el juego en su totalidad.

Se incluyen los subapartados *Starcraft 2* y *Aprendizaje por refuerzo* para una mayor contextualización.

1.1.1. Starcraft 2

Starcraft 2 (Blizzard Entertainment,2010)[2] es un juego de estrategia en tiempo real (RTS). Considerado por la mayoría (junto a su predecesor *Starcraft*) como uno de los videojuegos más difíciles de la historia, fue el precursor de todo el fenómeno de los *e-sports*.

Starcraft 2 dispone de millones de jugadores a lo largo de todo el mundo, con una activa escena competitiva de alto nivel en la que profesionales de élite, que entrenan un gran número de horas diarias, compiten en ligas y campeonatos. Esta escena consta de muchos canales en Twitch.tv dedicados a la retransmisión de estas competiciones e incluso canales de televisión dedicados como la OGN - un canal de televisión por cable de Corea del Sur. *Starcraft 2* es especialmente popular en Corea del Sur, donde emergió cómo *e-sport*. Los mejores jugadores coreanos han dominado la escena desde el principio y son considerados un paso por encima de los demás, siendo muy pocos los jugadores del resto del mundo (o *foreigners*, como se les conoce en la escena) los que han conseguido desbancarlos.

En el modo 1 contra 1 de *Starcraft 2* (en el que se centra toda la escena competitiva y también este proyecto), cada jugador controla una de las tres razas disponibles (Terran, Protoss y Zerg). El jugador empieza con un edificio principal (Centro de Mando, Nexo o Criadero, respectivamente) y una serie de recolectores (VCEs, Sondas o Zánganos). El objetivo final es acabar con el oponente. Para ello, el jugador deberá, a grandes rasgos, desarrollar su tecnología, construir edificios de producción, crear unidades, expandirse

por el mapa para poder obtener más recursos, y eventualmente vencer a su oponente en combate. La partida termina cuando un jugador destruye todos los edificios de su oponente o este se rinde. [3] [4]

La dificultad del juego radica en el gran número de factores que influyen; el número de estrategias posibles para cada raza, en cada mapa, en cada momento concreto; el control de las unidades; el hecho de que sea un juego de información parcial en el que el jugador sólo tiene información de lo que ven sus unidades o edificios y por tanto requiere de exploración constante para poder adaptarse a la estrategia de su oponente; y un largo etcétera.

A continuación se definen una serie de conceptos claves [3] [4]:

- **Recolección de recursos**: Consiste en obtener recursos para poder construir unidades, edificios o desarrollar tecnologías. En *Starcraft 2* existen dos recursos: mineral y gas vespeno.
- **Edificios**: Permiten generar unidades y desarrollar tecnologías. En ocasiones se construyen en posiciones determinadas para bloquear caminos, o crear estrechamientos para defender mejor. La partida termina cuando todos los edificios de un jugador son destruidos.
- **Orden de construcción** o **Build order**: Orden de construcción de unidades y edificios calculado de manera precisa para conseguir el inicio óptimo de una estrategia. Normalmente se marca el orden con tiempos (0:56 - Depósito de suministros) o con número de población(10/12 - Depósito de suministros).
- **Exploración** o **scouting**: Consiste en enviar unidades o utilizar otros métodos como el **Scan** de los Terran para conseguir información sobre alguna parte del mapa y deducir la estrategia del oponente. Esta obtención de información es tal vez la parte más importante de *Starcraft 2*.
- **Microcontrol** o **micro**: Control preciso y efectivo de las unidades. Ataques enfocados, separación rápida de unidades para evitar daño de área, uso efectivo de las habilidades, hit and run...
- **Macrocontrol** o **macro**: Administración efectiva de la economía. Producción constante de unidades, gasto de los recursos sin dejar que estos

se acumulen, investigaciones tecnológicas, mantenimiento del límite de población siempre por encima de la población actual...

- **Multitasking**: La capacidad de realizar múltiples acciones a la vez como múltiples ataques por varios flancos. Esta habilidad es esencial y diferencia a los mejores jugadores, ya que aquellos con un multitasking refinado pueden manejar perfectamente su economía(macro) a la vez que controlan de manera eficaz potentes ataques por varios flancos (micro).

1.1.2. Aprendizaje por refuerzo

Se trata de un enfoque de Machine Learning inspirado en la manera en que humanos y animales aprenden a tomar decisiones mediante recompensas positivas o negativas. En un **momento** concreto t, el agente se encuentra en un **estado** s y decide realizar una **acción** a de entre todas las acciones disponibles en su estado actual. Cómo respuesta, el entorno le otorga una **recompensa** inmediata r. El agente va a aprendiendo a seleccionar las acciones que maximizan su suma de recompensas.

Más formalmente, el objetivo del agente es descubrir una política para seleccionar acciones que maximicen una medida de recompensa a largo plazo. Las interacciones con el entorno se modelan como un Problema de Decisión de Markov (MDP), definido por un conjunto de estados, un conjunto de acciones posibles en cada estado, la probabilidad de transición entre dos estados dada una acción y la función de recompensa al pasar de un estado a otro dada una acción[1][5] [6].

1.2. Actores implicados

1.2.1. Desarrollador del proyecto

El rol de desarrollador del proyecto consiste en la realización de todas las tareas asociadas al mismo (programación, tests, estudios y documenta-

ción). Además es responsable de cumplir los plazos y seguir el horario y la planificación acordada con el director, así cómo de mantener una buena comunicación con este. Este proyecto dispone de un único desarrollador, Carlos Roldán Montaner, estudiante de último cuatrimestre de ingeniería informática en la Facultad de Informática de Barcelona (UPC).

1.2.2. Director del proyecto

El director de proyecto es el encargado de guiar y asistir al desarrollador. Su trabajo consiste en dar consejo, opinión y ayudar al desarrollador con el enfoque y ejecución del proyecto, contribuyendo a solventar posibles obstáculos. El director de este proyecto es Javier Béjar Alonso, profesor titular del departamento de Ciencias de la Computación en la Facultad de Informática de Barcelona (UPC) y miembro del Knowledge Engineering and Machine Learning Group (KEMLg) como especialista en Machine Learning.

1.2.3. Beneficiarios

Debido al carácter de investigación de este proyecto, es difícil nombrar beneficiarios directos. No se trata de un producto o nueva tecnología con un efecto directo e inmediato sobre las personas. Sin embargo, como se ha mencionado, *Starcraft 2* es uno de los grandes retos de la IA. Los avances en cuanto a dominar este juego pueden ser, por tanto, beneficiosos para todo el sector de la IA. Averiguar qué enfoques y algoritmos pueden contribuir a solucionar un problema de la complejidad que presenta *Starcraft 2* puede ser un objetivo de gran utilidad.

Como beneficiarios directos, podemos encontrar a los jugadores, en especial los profesionales, que dispondrían de una herramienta única con la que poder entrenar y perfeccionar sus habilidades.

1.3. Estado del arte

El aprendizaje por refuerzo está obteniendo cada vez mejores resultados. DeepMind[7], la compañía adquirida por Alphabet ha conseguido ya un nivel superhumano en muchos videojuegos, siendo AlphaGo, capaz de vencer al campeón del mundo de Go, su último logro. Existen numerosos algoritmos de aprendizaje por refuerzo, pero los nuevos avances se están centrando en mezclarlos con Deep Learning, convirtiéndose así en Deep Reinforcement Learning [8] (con el que se experimentará en este proyecto). Algunos usos destacados, además de AlphaGo, son el proyecto conjunto de Google y DeepMind para hacer más eficientes sus *data centers* o aún más destacado, en los coches autónomos, con los que ya se harán pruebas reales a final de este año en un proyecto conjunto de BMW, Mobileye e Intel. Google y Uber también lo están aplicando a sus vehículos autónomos[9].

En cuanto a la IA en *Starcraft*, existen competiciones donde distintos bots compiten entre si. Las más importantes son SSCAIT (*Student Starcraft AI Tournament*), en la que las IAs compiten online en dos fases, un torneo de tres semanas, y una clasificación que dura 11 meses, en la que los espectadores pueden utilizar un sistema de voto online para decidir que bots se enfrentarán en el siguiente combate; AIIDE (*Artificial Intelligence and Interactive Digital Entertainment*), una competición anual en la que los bots compiten 24 horas al día durante 2 semanas, con el requisito de que todos los códigos sean open source y se publiquen al terminar la competición; y CIG (*Computational Intelligence in Games*), en la que los códigos no tienen porque ser *open source* y a diferencia de las otras dos, los mapas no son conocidos con anterioridad.

Aunque muchos de estos bots utilizan simples scripts, sistemas de reglas, o máquinas de estados, en los últimos años han aparecido algunos que se sirven de técnicas más avanzadas como algoritmos genéticos, redes neuronales y otras formas de aprendizaje supervisado y algunas formas de aprendizaje por refuerzo. Aunque estos bots no pueden alcanzar el nivel de un humano, algunos de ellos son capaces de generar buenas composiciones de unidades, expandirse y gestionar recursos y unidades de manera decente [10].

En cuanto a Starcraft 2, aún no existen muchos avances. Existe un *ladder* en sc2ai.net en el que los bots pueden competir contra otros, pero todavía

hay muy pocos y son *scripted*, con un aprendizaje nulo o muy limitado.

1.3.1. Los agentes de DeepMind y el enfoque de este proyecto

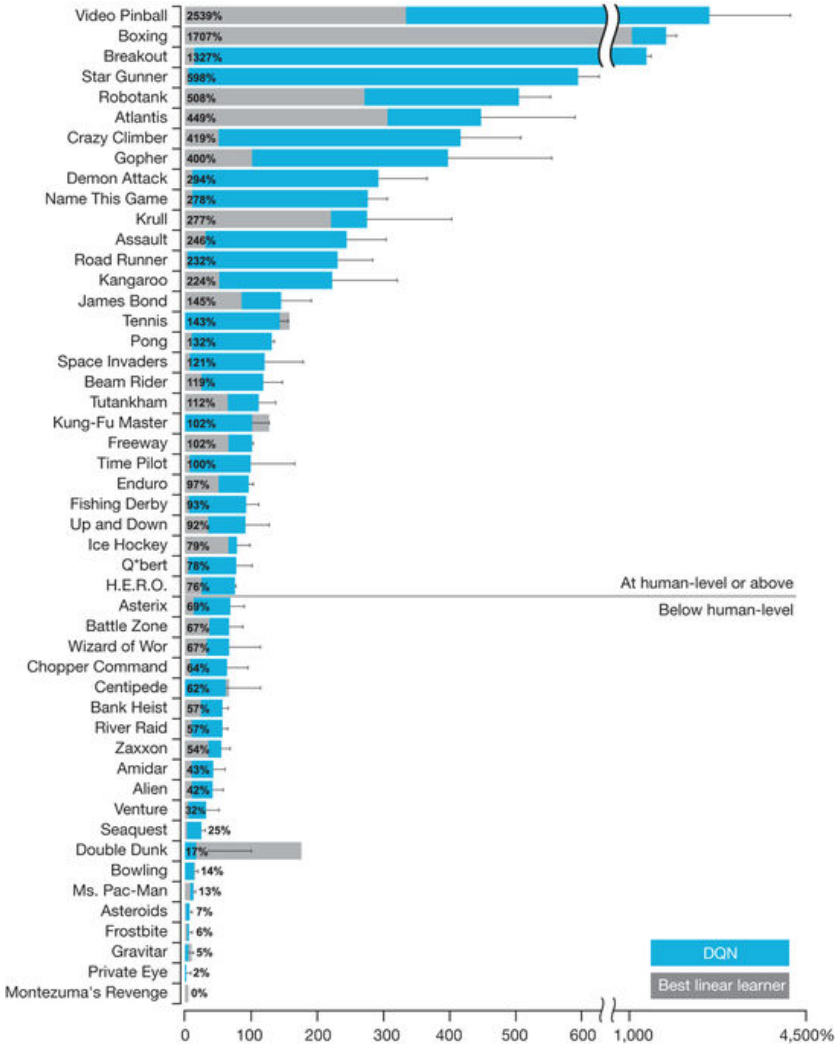

Figura 1.1: Nivel obtenido por los agentes de DeepMind en los juegos de Atari

DeepMind han sido los pioneros en Deep Reinforcement Learning. En 2015 lanzaron su algoritmo de Deep Q-Network (DQN), capaz de aprender a jugar a un amplio rango de juegos de Atari 2600 por su cuenta, con solo la información de los píxeles de la pantalla como input. Más adelante, y después de alguna mejora a este algoritmo, lanzaron su agente *Asynchronous Advantage Actor-Critic* o A3C. Este es actualmente el algoritmo más poderoso de aprendizaje por refuerzo actual, (aunque queda fuera del alcance de este proyecto por limitaciones computacionales y de tiempo).

Aunque conseguir agentes que dominen los juegos de Atari 2600 de manera sobrehumana partiendo de cero es sin duda impresionante, es importante recalcar una característica de estos juegos: son relativamente simples. Más específicamente, su espacio de acciones es muy limitado. En *Starcraft 2* en cambio, es inmenso. Esto explica que los agentes de DeepMind que son capaces de dominar juegos enteros de Atari, fracasen por complejo en escenarios simples de *Starcraft 2*. Para simplificar: es más sencillo saber que acción es mejor tomar entre 4 acciones posibles, que entre un millón.

Como se puede observar en la tabla anterior, los agentes de DeepMind solo igualan el nivel de juego humano en el caso más simple, mientras para los últimos escenarios más complejos (que a nivel de juego son bastante básicos), no son capaces de aprender ni obtener resultados satisfactorios.

DeepMind utiliza un input enorme (toda la información de la pantalla representada en píxeles) y su mejor algoritmo. Esto requiere una cantidad de recursos elevada de la que este proyecto no dispone. Sin embargo, sería ingenuo pensar que utilizar el mismo enfoque con algoritmos más sencillos computacionalmente puede dar resultados similares. Por esto, el enfoque establecido es otro.

AGENT	METRIC	MOVETOBEACON	COLLECTMINERALSHARDS	DEFEATROACHES	DEFEATZERGLINGSANDBANELINGS	COLLECTMINERALSANDGAS	BUILDMARINES
Random Policy	MEAN	1	17	1	23	12	<1
	MAX	6	35	46	118	750	5
Random Search	MEAN	25	32	51	55	2318	8
	MAX	29	57	241	159	3940	46
DeepMind Human Player	MEAN	26	133	41	729	6880	138
	MAX	28	142	81	757	6952	142
Starcraft GrandMaster	MEAN	28	177	215	727	7566	133
	MAX	28	179	363	848	7566	133
Atari-Net	BEST MEAN	25	96	101	81	3356	<1
	MAX	33	131	351	352	3505	20
FullyConv	BEST MEAN	26	103	100	62	3978	3
	MAX	45	134	355	251	4130	42
FullyConv LSTM	BEST MEAN	26	104	98	96	3351	6
	MAX	35	137	373	444	3995	62

Figura 1.2: Resultados obtenidos por DeepMind.

En lugar de tomar como estado todos los píxeles, se definen los estados a partir de los parámetros que tengan sentido en cada escenario. Así se elimina información redundante (un gran porcentaje de los píxeles no cambia en ningún momento en estos escenarios, por ejemplo).

En cuanto a las acciones, se eliminan las acciones imposibles (seleccionar unidades de una raza que no está en el escenario sería un ejemplo) y las

que no tengan sentido. Además, se definen acciones con semántica (que se detallarán en la sección 7.2.1). Es decir, se aplica ingeniería del conocimiento.

DeepMind propone un aprendizaje desde cero. Este proyecto propone un aprendizaje lógico. Donde DeepMind intenta enseñar a jugar a un bebé recién nacido, este proyecto intenta enseñar a tu amigo al que le gustan los videojuegos pero nunca ha jugado a *Starcraft 2*. Se han ido avanzando desde algoritmos más simples a más complejos y se han evaluado los resultados.

1.4. Alcance del proyecto

El primer paso fue la instalación y configuración de PySC2, una colaboración de Google DeepMind y Blizzard Entertainment, que utiliza la API de machine learning de Blizzard para *Starcraft 2* en un entorno de Python para aprendizaje por refuerzo.

Una vez realizado este paso, se eligieron distintos escenarios en los cuales entrenar los agentes. Estos escenarios son concretos y de variada dificultad. El objetivo fue recrear distintos escenarios típicos de una partida real y que además requieren distintos tipos de control y jugabilidad. Por ejemplo, escenarios de combate entre distintos tipos y cantidades de unidades (que requieran distintos tipos de microcontrol), escenarios de recolección de recursos, de construcción de edificios, etc.

Se empezó por escenarios sencillos y se fue incrementando la dificultad. Para cada escenario se entrenaron agentes con distintos métodos de aprendizaje por refuerzo (Q-Learning, SARSA, DQN...) y se estudiaron los resultados obtenidos con distintas funciones de refuerzo para determinar cuales obtienen mejores rendimientos. El resultado ideal sería una IA competente en estos escenarios específicos.

1.5. Análisis de alternativas

Desde el inicio y a lo largo del proyecto han surgido numerosas situaciones en las que han tenido que tomarse decisiones y buscar alternativas para superar distintos problemas y dificultades.

Antes de explicarlas, es lógico ofrecer un poco de contexto. El objetivo del proyecto es desarrollar agentes de IA para Starcraft 2. Debido a la complejidad del juego, conseguir agentes que dominen el juego es un propósito alejado de la realidad. En su lugar, el ámbito del proyecto se enfoca en desarrollar agentes para escenarios específicos que involucran comportamientos comunes dentro del juego.

La primera decisión fue escoger el tipo de aprendizaje que realizarían los agentes. Se escogió aprendizaje por refuerzo en lugar de otras alternativas como por ejemplo aprendizaje supervisado debido a varios motivos. Uno de ellos es que es el enfoque escogido por DeepMind. Sus agentes utilizan algoritmos de vanguardia de aprendizaje por refuerzo, de manera que son un buen *benchmark* con el que comparar. Otro motivo importante es que para entrenar agentes de aprendizaje supervisado, se requerirían muchísimas repeticiones de partidas reales. Aunque estas existen, y han sido publicadas por Blizzard, se trata de repeticiones sobre partidas completas, que como se ha mencionado anteriormente se escapa del alcance de este proyecto.

A lo largo del proyecto ha sido necesario valorar distintas alternativas en muchas situaciones: desde los algoritmos (o variantes de ellos a utilizar); los escenarios a entrenar en función de la dificultad, el tiempo disponible y el éxito de los anteriores; la manera de definir el espacio de acciones y estados para cada uno de ellos; etc. Todas estas alternativas han sido valoradas y en varios casos se han hecho pruebas con varias de ellas, como se comentará en profundidad en capítulos posteriores.

1.6. Metodología

Para este proyecto se han aplicado técnicas y características propias de las metodologías Agile. Se ha trabajado en ciclos cortos, cada uno de los cuales se ha basado en un escenario, para el que se han entrenado distintos agentes, evaluado sus respectivos rendimientos y sacado conclusiones que se han aplicado a los siguientes escenarios en próximas iteraciones.

También se han organizado reuniones semanales o bisemanales con el director para evaluar cada iteración y obtener *feedback* constante, facilitando así la resolución de obstáculos y dificultades y garantizando un buen ajuste a la planificación establecida.

1.7. Herramientas

1.7.1. Herramientas de desarrollo

Para el desarrollo del proyecto se han utilizado las siguientes herramientas:

- **PyCharm**: El proyecto ha sido desarrollado en Python, haciendo uso del framework PySC2. Para poder visualizar la interfaz de "feature layers"que proporciona el framework es necesario un IDE. En este caso se ha usado PyCharm, de JetBrains, aunque también podría usarse IntelliJ con plugins para Python.
- **LaTeX**: La documentación de este proyecto ha sido llevada a cabo en LaTeX. Se ha elegido esta herramienta por el gran número de opciones que ofrece y los resultados tan limpios que produce, ante alternativas como Microsoft Word o Google Docs.

1.7.2. Herramientas de seguimiento

Para el seguimiento y monitorización del proyecto se han usado las siguientes herramientas:

- **Git y GitHub**: Se utilizarán Git y GitHub como repositorio para el proyecto, así cómo para llevar un seguimiento de la evolución del mismo.
- **Trello**: Trello permite organizar de manera clara y ordenada las tareas y facilita seguir la planificación correctamente.

1.8. Métodos de validación

La validación se ha realizado a partir de los objetivos de cada escenario.

Adicionalmente, podrían haberse incluirse valoraciones respecto a la naturalidad de los movimientos de cada algoritmo. Así tendríamos dos medidas, comprobando como de bueno es el juego de la IA y cómo de creíble es (es decir, si es similar a cómo jugaría una persona).

1.9. Integración de conocimientos

Al ser un proyecto de inteligencia artificial, los conocimientos utilizados han sido esencialmente de esta disciplina. Sin embargo, también han estado presentes conocimientos de varios ámbitos de computación, algoritmia, clustering, etc., así como conceptos generales de programación.

Se han propuesto soluciones distintas a problemas que los agentes no podían superar, mediante ingeniería del conocimiento. También se han propuesto soluciones para habilitar distintas acciones lógicas a la vez que limitando el número de acciones posibles, como la división del mapa de juego en sectores.

1.10. Posibles problemas y obstáculos

1.10.1. La dificultad del problema

Conseguir que la IA consiga buenos resultados no es tarea fácil, y menos en *Starcraft 2*. Ha sido necesaria una selección de algoritmos y muchas pruebas y ajustes para conseguirlo. Aún así, no todo han sido éxitos, como se verá en capítulos posteriores.

1.10.2. El tiempo y la planificación

Los cuatro meses disponibles para la realización del proyecto son bastante restrictivos. Aunque a priori parecía posible conseguir resultados positivos, esto ha requerido un ajuste estricto a la planificación. Además, entrenar agentes es un proceso lento que ha consumido muchas horas de computación, especialmente cuando había que entrenar un agente numerosas veces para ajustar parámetros e intentar mejorar sus resultados, lo que nos lleva al siguiente problema.

1.10.3. La potencia de computación

La potencia de computación requerida por técnicas de este estilo es elevada. Para paliar este problema se ha utilizado una GPU NVIDIA GTX 960. Se planteó originalmente utilizar servicios de computación en la nube, pero finalmente se declinó debido a la problemática de instalar todo el entorno en los mismos y a los costes añadidos que habría supuesto.

Capítulo 2

Planificación del proyecto

2.1. Planificación y tiempos

El proyecto ha tenido una duración aproximada de unos 4 meses, dando inicio a mediados de febrero y terminando a finales de junio, antes de las exposiciones.

Dado que se trata de un proyecto de investigación en el cual ha sido necesario llevar a cabo experimentos diversos, no ha habido una planificación fija. De hecho, ha sido a partir de los resultados de los primeros experimentos que se ha determinado el rumbo a seguir. No obstante, si que se estableció una planificación general inicial.

Además, el uso de metodologías ágiles lleva implícito estas modificaciones de planificación.

2.2. Descripción de las tareas

2.2.1. Aprender sobre aprendizaje por refuerzo

A lo largo del mes de febrero, el primer objetivo fue obtener más conocimientos sobre aprendizaje por refuerzo, y familiarizarme con algunos conceptos. Para ello, primero leí unos cuantos artículos que trataban conceptos generales, como por ejemplo el de Deepmind [8]. A continuación, leí el libro *Artificial Intelligence and Games*, recomendado por mi tutor, en el que no solo obtuve información sobre aprendizaje por refuerzo, sino que aprendí mucho sobre IA en general y me sirvió para refrescar algunos conceptos.

Para implementar los algoritmos y ampliar mi conocimiento del aprendizaje por refuerzo, se utilizó un repositorio de GitHub [6], así como el curso de Google Deepmind [11] y el libro *Reinforcement Learning: An Introduction*[12].

2.2.2. Instalar y configurar el entorno

El primer paso fue realizar el setup del entorno de desarrollo. En primer lugar fue necesario instalar *Starcraft 2*, que puede obtenerse de manera gratuita. En este caso disponemos de la versión completa. Una vez hecho esto, se instaló todo el *framework* de PySC2. Fue necesario utilizar la versión 3.5 de Python ya que PySC2 requiere de Tensorflow, entre otras dependencias. En este proyecto el desarrollo se ha llevado a cabo en Windows. Además, se ha instalado todo lo necesario para poder ejecutar el código desde la GPU en caso de que fuese necesario.

También se han utilizado otras librerías auxiliares como NumPy y Pandas, la primera matemática y la segunda sobre Machine Learning.

2.2.3. Familizarizarse con el framework

PySC2 es un *framework* bastante grande y complejo. El juego se representa sobre una serie de *feature layers* o capas características. Para acceder a los parámetros del juego y poder consultarlos y manipularlos fue necesario familiarizarse primero con todo el entorno. La documentación es correcta pero no excepcional, así que se ha llevado a cabo un proceso de experimentación con el *framework*.

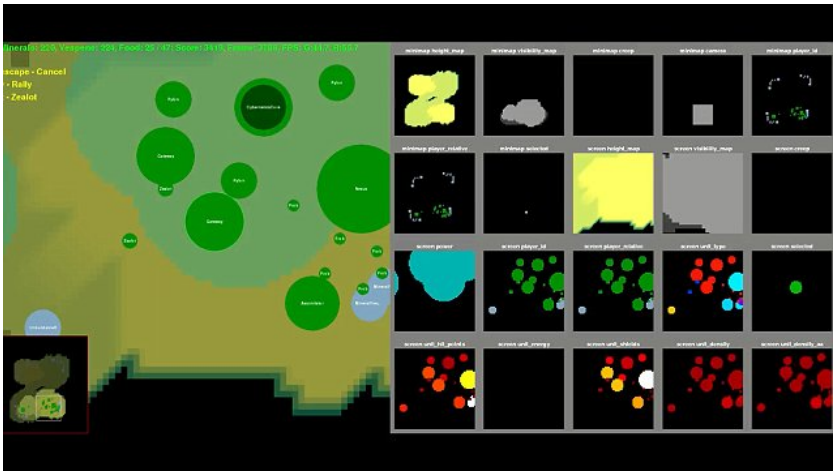

Figura 2.1: Feature layers de PySC2 para representar el entorno de *Starcraft 2*

Describir el entorno entero sería demasiado extenso y de poca utilidad. Sin embargo, caben destacar algunas características importantes. Una de ellas es la posibilidad de acelerar hasta cierto punto el juego, lo que reduce los tiempos de entrenamiento (aunque estos siguen siendo muy grandes).

En cuanto a las APM (acciones por minuto), están limitadas. Un jugador de nivel medio puede rondar las 120, mientras que los mejores profesionales oscilan las 400. Naturalmente, una inteligencia artificial tiene muchísima ventaja en este aspecto.

En cuanto a las observaciones, PYSC2 hace un buen trabajo. Además de los píxeles RGBs (en los que se dispone de toda la información, pero es muy

complicada de extraer), dispone de diversas capas de características (*feature layers* que ofrecen información que no se espera que los agentes aprendan a leer por si solos. Por ejemplo, dos de ellas son **Minimap** y **Screen**, que ofrecen información sobre los niveles del terrero, las partes visibles, las unidades, la salud, etc.

Además, otras capas ofrecen información sobre las unidades seleccionadas, las acciones disponibles, o la información general que se muestra en pantalla, como los minerales, el gas vespeno, la población usada y máxima, etc.

Es importante destacar que DeepMind plantea así un entorno en el que la información obtenida es la misma (o al menos esta es la idea) que la que inferiría un jugador que observase la pantalla. Esto contrasta con otros enfoques como los de CommandCenter (o TorchCraft o BWAPI para Starcraft), en las que se dispone de información adicional a nivel más bajo. Por ejemplo, con la API de PySC2, no podemos obtener un identificador para cada unidad individual (ya que un jugador no lo tendría). Esto dificulta mucho las tareas de microcontrol, ya que evaluar la salud de una unidad o realizar cierto comportamiento con ella se complica, sobretodo porque no podemos distinguir esa unidad de otra. Esto nos lleva a alternativas tal vez menos elegantes (aunque tal vez más humanas) para tareas como seleccionar distintas unidades y también a alternativas más lógicas o de más alto nivel, como se discutirá más adelante en el apartado 7, así como en los apartados dedicados a los agentes.

Respecto al espacio de acciones, es inmenso. Del orden de millones o tal vez billones. Muchas de estas acciones son inválidas en un momento determinado, y otras tienen una extrema correlación entre ellas. PySC2 intenta reducirlas a acciones más ricas semánticamente, y establece un conjunto de acciones basado en las acciones que humanos han realizado en el pasado, extraídas de un gran número de repeticiones de partidas. Finalmente, el conjunto de acciones se ve reducido a bastantes centenares de acciones. Que son más, muchas más, ya que hay acciones que requieren un input espacial. Por ejemplo, la acción **"Mover"** requiere una coordenada x y una coordenada y. De esta manera, suponiendo una resolución reducida de 64*64 para simplificar, una acción se convierte en 1 x 64 x 64 acciones. Y así con todas.

Para una compañía con los recursos de Google detrás tal vez esto sea

suficientemente pequeño. Para ejemplificarlo aún más, cada uno de sus entrenamientos para los agentes de Atari (muchísimo más simples) era de 38 días. Como este proyecto no puede permitirse eso, se limitarán las acciones a aquellas que se consideren lógicas para cada escenario. Se profundizará más sobre este tema en el apartado 7, así como en los apartados dedicados a los agentes.

Para empezar, se crearon los primeros agentes. Estos eran bots simples, *scripted*, siguiendo un modelo de máquina de estados. Una vez estaban funcionando, se podía iniciar el desarrollo con modelos de aprendizaje por refuerzo.

2.2.4. Selección inicial de escenarios

Como se ha mencionado con anterioridad, este proyecto se ha centrado en desarrollar agentes que aprendan a resolver satisfactoriamente un conjunto de escenarios concretos de *Starcraft 2*. Por tanto, es necesario definir cuales serán estos escenarios. Es importante remarcar que esta definición de escenarios no era definitiva. Los agentes fueron entrenados primero en escenarios de menor dificultad con diversos algoritmos, y en función del rendimiento obtenido se modificaron.

Los escenarios se plantearon de manera que incluyen tareas tan diversas como sea posible. De esta manera podemos observar el comportamiento de la IA ante diferentes tipos de problema. Por ejemplo podría resultar que los agentes obtienen buenos resultados en escenarios de combate, donde el microcontrol de las unidades es lo más importante, y en cambio obtener un rendimiento pobre en escenarios donde la prioridad es la construcción o gestión de recursos.

Además, desarrollar agentes eficaces en distintos ámbitos facilitaría la creación de una IA que dominase los distintos aspectos del juego, aunque eso está fuera del alcance de este proyecto.

En caso de haber necesitado nuevos escenarios, estos habrían sido generados usando la herramienta Editor de Mapas de *Starcraft 2*, aunque esto finalmente no fue necesario.

2.2.5. Implementación, ejecución y *testing* de los agentes

La parte tal vez más importante es la creación de la IA en sí. Se implementaron agentes específicos a cada escenario utilizando una selección de algoritmos de aprendizaje por refuerzo. Se realizaron distintas ejecuciones intentando ajustar los parámetros de cada agente para obtener resultados óptimos. Se llevaron a cabo pruebas para determinar el correcto funcionamiento de un agente, y se compararon sus resultados con otros agentes sobre el mismo escenario pero implementados con distintos métodos. Dependiendo del tiempo y los resultados se utilizaron unos algoritmos u otros.

Esta parte fue la más costosa debido a que implementar agentes de *reinforcement learning* para *Starcraft 2* no es tarea trivial, pero sobretodo a los problemas generados por el coste computacional.

2.2.6. Análisis y conclusiones

Se analizaron los resultados de los distintos algoritmos en diferentes escenarios y se sacaron conclusiones sobre ellos, comparando el rendimiento de cada uno con el resto para comprobar cuáles responden mejor.

2.3. Plan de acción y posibles retrasos

El plan de acción para este proyecto consistió básicamente en llevar a cabo en orden las tareas propuestas anteriormente. Gracias a las metodologías ágiles y al *feedback* por parte del director, se fueron modificando la planificación y los experimentos. En caso de que una tarea terminase antes de tiempo se empezaba inmediatamente la siguiente. Los resultados obtenidos hacían variar el plan de acción. Por ejemplo, si un algoritmo obtenía resultados muy pobres, se dedicaba más tiempo a otros más productivos.

Las principales causas de retrasos (al margen de las ajenas al proyecto), fueron los elevados tiempos de computación en los entrenamientos, el proceso

de ajuste de parámetros y la implementación de los algoritmos.

2.4. Diagrama de Gantt inicial

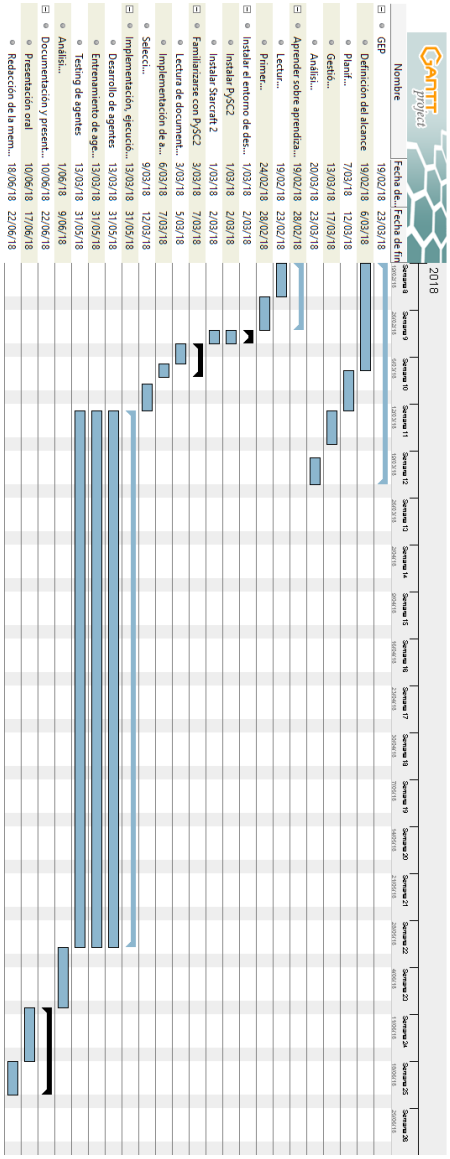

Figura 2.2: Diagrama de Gantt inicial

El motivo de la ausencia de dependencias en el Gantt es debido tanto a limitación de software como a la naturaleza del proyecto. Por ejemplo, todo el desarrollo depende de haber adquirido conocimiento mediante la documentación, pero añadirlo como dependencia directa haría que el desarrollo empezase justo después de leer documentación, cosa que no es cierto. Y una cadena de dependencias tampoco funcionaría, ya que instalar el entorno no depende de haber leído documentación.

Es necesario remarcar de nuevo que toda esta estimación es una aproximación general. Dependiendo de la velocidad con que tengan éxito los algoritmos, el tiempo de ejecución de los mismos, el número de veces que haya que lanzar cada uno... la planificación se verá muy modificada. Además, las tareas de desarrollo están en paralelo ya que para cada algoritmo y escenario se hará desarrollo, entrenamiento y testing.

2.5. Planificación final, evaluación y cambios

La planificación se ha seguido según lo planeado originalmente. Como cabía esperar, el desarrollo y entrenamiento de los agentes (en especial esto último) ha sido lo que más tiempo ha consumido, especialmente con la implementación de una Deep-Q Network con Experience Replay (de la que se hablará en apartados posteriores).

El hecho de tener que entrenar agentes varias veces, debido a resultados insatisfactorios, ajuste de parámetros, redefinición del espacio de acciones o estados, o simplemente *bugs* ha incrementado ligeramente este tiempo. Destacar en especial el cuarto agente, *DefeatZerglingsAndBanelings*, que ha resultado especialmente difícil de entrenar.

Los objetivos del proyecto no se han visto perjudicados por la planificación. Se han entrenado agentes de aprendizaje por refuerzo con distintos algoritmos, y se han comparado los resultados obtenidos. En general, con alguna excepción, estos han sido satisfactorios; tanto como se podría esperar de un proyecto con recursos limitados.

También, la adquisición de conocimientos sobre aprendizaje por refuerzo,

en lugar de hacerse al principio, se ha extendido a lo largo del proyecto. Se mostrará en el Gantt final como una tarea continua, pero no implica un número tan elevado de horas como el de entrenar agentes, sino indica que durante el proceso de entrenar agentes se han ido adquiriendo más conocimientos.

A su vez, el análisis de resultados no ha incrementado en número de horas pero se ha ido llevando a cabo a medida que se iban entrenando agentes para facilitar el desarrollo de los siguientes mediante las conclusiones extraídas.

Finalmente no se incluyeron nuevos escenarios, porque añadir escenarios de menor complejidad sería redundante y debido a las dificultades de la IA con los escenarios complejos, no había expectativas de que pudiese progresar más allá. Además, los costes temporales habrían superado el número de horas designadas al proyecto, especialmente debido a los entrenamientos.

2.6. Diagrama de Gantt final

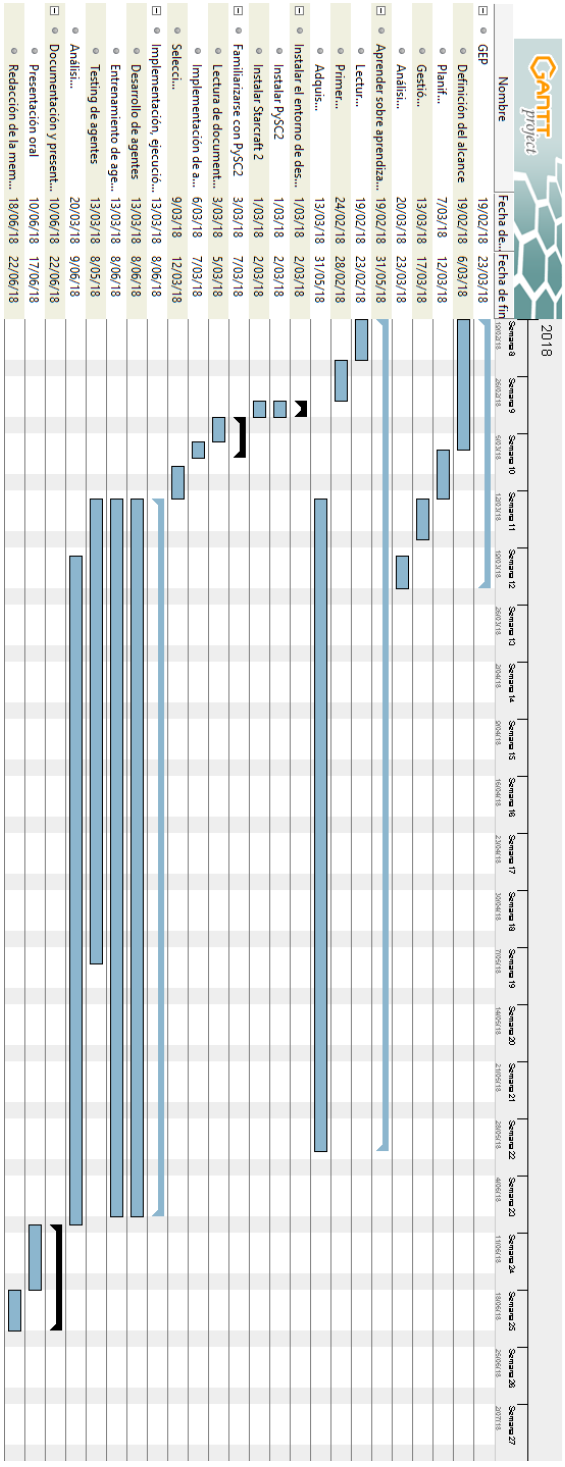

Figura 2.3: Diagrama de Gantt final

Capítulo 3

Coste y sostenibilidad

3.1. Coste del proyecto

Para llevar a cabo este proyecto, fueron necesarios los recursos mencionados en los apartados anteriores. A continuación se muestra una estimación general de los costes, teniendo en cuenta los gastos de hardware, software y los costes indirectos que puedan aparecer.

Tarea	**Tiempo estimado (h)**
GEP	75
Aprender sobre aprendizaje por refuerzo	60
Instalar y configurar el entorno	5
Familiarizarse con PySC2	20
Selección y diseño inicial de escenarios	10
Implementación, ejecución y testing de los agentes	300
Análisis y conclusiones	28
Redacción de la memoria	30
Presentación oral	12
Total	540

Cuadro 3.1: Estimación de tiempo por tareas

3.1.1. Presupuesto hardware

A continuación se detallan los costes del hardware utilizado en este proyecto, así como la amortización estimada para cada uno.

Producto	**Precio**	**Unidades**	**Vida útil**	**Amortización**
PC	1000€	1	4 años	56.81€
Monitor Philips 17"	80€	2	5 años	4€
Total	1160€			64.81€

Cuadro 3.2: Estimación de costes hardware

3.1.2. Presupuesto del software

Producto	**Precio**	**Unidades**	**Vida útil**	**Amortización**
PyCharm	0€	1	-	0€
Starcraft II	0€	1	-	0€
GanttProject	0€	1	-	0€
GitHub	0€	1	-	0€
ShareLatex	0€	1	-	0€
Trello	0€	1	-	0€
Total	0€			

Cuadro 3.3: Estimación de costes software

En este caso, el presupuesto de software es 0 euros ya que todos los programas utilizados son gratuitos. En el caso de *PyCharm*, IDE de JetBrains, se dispone de una licencia de estudiante. En el caso de *Starcraft II*, Blizzard lo lanzó recientemente de manera gratuita.

3.1.3. Estimación de costes humanos

Este proyecto está desarrollado por una sola persona. Por tanto los gastos de recursos humanos se obtendrán mediante el total de horas de trabajo

multiplicadas por el salario estimado por hora.

Ya que la persona responsable del proyecto ha tenido que realizar roles de desarrollo, gestión y *tester*, se estima un salario de 40€ la hora.

Tarea	**Tiempo estimado (h)**
GEP	75
Aprender sobre aprendizaje por refuerzo	60
Instalar y configurar el entorno	5
Familiarizarse con PySC2	20
Selección y diseño inicial de escenarios	10
Implementación, ejecución y testing de los agentes	300
Análisis y conclusiones	28
Redacción de la memoria	30
Presentación oral	12
Total	540

Cuadro 3.4: Estimación de tiempo por tareas

La tabla anterior muestra una estimación de las horas por tarea. Teniendo en cuenta un salario de 40€ la hora, el gasto humano total habría sido de unos 21600€.

Tarea	**Tiempo estimado (h)**
GEP	3000€
Aprender sobre aprendizaje por refuerzo	2400€
Instalar y configurar el entorno	200€
Familiarizarse con PySC2	800€
Selección y diseño inicial de escenarios	400€
Implementación, ejecución y testing de los agentes	12000€
Análisis y conclusiones	1120€
Redacción de la memoria	1200€
Presentación oral	480€
Total	21600€

Cuadro 3.5: Estimación de costes por tareas

3.1.4. Costes imprevistos

En un proyecto de este tipo, es difícil realizar una estimación precisa de los costes. Existen muchos factores que pueden hacer que estos incrementen.

En este caso, se tienen en cuenta dos principales:

- **Incremento del número de horas**: En proyectos de Machine Learning, especialmente los que cuentan con pocos recursos de hardware como es el caso de este, las horas pueden verse drásticamente incrementadas debido a los tiempos de entrenamiento de los agentes.
- **Uso de otros recursos hardware**: Aunque a priori no parece necesario, tener que hacer uso de sistemas de computación en la nube incrementaría los costes de hardware.

El segundo no se ha tenido en cuenta, ya que finalmente no se ha optado por esa opción. Sin embargo, para el primer paso, se ha dado un margen de 50 horas que supondría 2000 euros de costes inesperados.

3.1.5. Costes indirectos

Producto	**Precio**	**Unidades**	**Coste Estimado**
Electricidad	0.14kW€	1500kWh	210€
Internet	30€/mes	4 meses	120€
Espacio	0€	-	0€
Otros	100€	-	100€
Total			430€

Cuadro 3.6: Costes indirectos

La tabla anterior muestra una estimación de los costes indirectos del proyecto. Costes como material de oficina, transporte, uso de otros ordenadores... se engloban en el apartado *Otros*. Los costes de espacio son nulos ya que el proyecto se ha llevado a cabo en el domicilio familiar.

3.1.6. Costes totales

Concepto	**Coste estimado**
Costes Hardware	1160€
Costes Software	0€
Costes Humanos	21600€
Costes Imprevistos	2000€
Costes Indirectos	430€
Total	25190€
Total + Contingencia 10 %	27709€

Cuadro 3.7: Costes totales

En la tabla anterior pueden observarse los costes totales, así como la aplicación sobre ellos de una contingencia del 10 por ciento.

3.2. Control de presupuesto

Para el control de presupuesto, al final de cada tarea se anotarán el tiempo de horas real y el coste real. Se calculará la diferencia entre las horas reales y estimadas y se multiplicará por el coste estimado para calcular la desviación de consumo. También se calculará la diferencia entre el coste real y el estimado y se multiplicará por el número de horas reales para calcular la desviación de coste.

Mediante este proceso será sencillo observar las desviaciones ocurridas y determinar si es debido a un aumento del número de horas o a un incremento de los costes.

3.3. Sostenibilidad e impacto social

Debido al carácter de investigación del proyecto, es difícil realizar un análisis de sostenibilidad e impacto social, ya que no se trata de ningún producto o servicio, sino de una investigación sobre IA y Machine Learning. Aún así, se ha intentado responder a las preguntas que tengan sentido para este proyecto.

	PPP	**Vida Útil**	**Riesgos**
Ambiental	5/10	7/10	-
Económico	9/10	9/10	-
Social	9/10	9/10	-

Cuadro 3.8: Matriz de sostenibilidad

3.3.1. Dimensión ambiental

El impacto ambiental de este proyecto es mínimo, ya que el único recurso con un impacto medioambiental es la electricidad utilizada en mantener el equipo funcionando. Esto a su vez limita mucho la reutilización de recursos. Por tanto, un amplio estudio del impacto medioambiental no tiene sentido para este proyecto.

3.3.2. Dimensión económica

En cuanto a la dimensión económica, se ha llevado a cabo un estudio detallado de los costes implicados en este proyecto.

El estudio propuesto será más económico que los estudios llevados a cabo anteriormente (por ejemplo el de DeepMind), simplemente debido a DeepMind tiene muchos más recursos computacionales, que conllevan más gastos (y mayor impacto medioambiental).

3.3.3. Dimensión social

Los beneficiarios de este proyecto serían otros investigadores en temas de inteligencia artificial, machine learning y específicamente aprendizaje por refuerzo, que tendrían más datos de algoritmos y escenarios con los que realizar sus estudios y resultados.

A nivel personal, es una manera de ampliar mis conocimientos en el tema. Estudiar aprendizaje por refuerzo en *Starcraft 2* podría servir para desarrollar nuevas soluciones basadas en machine learning, por lo que sí, existe una necesidad de carácter de investigación.

3.4. Encuesta de sostenibilidad

Tras haber realizado la encuesta, me he dado cuenta de que la sostenibilidad no suele tenerse muy en cuenta. En los proyectos en los que he estado y en las empresas en las que he trabajado, la sostenibilidad desde luego que no ha sido una prioridad y en muchos casos, ni siquiera ha estado presente. Al contrario, el objetivo ha sido el éxito del producto y la maximización de los beneficios, sin tener mucho en cuenta ni el impacto social ni el medioambiental.

A la hora de evaluar mis conocimientos, creo que son bastante competentes en cuanto a la evaluación económica, ergonómica, de accesibilidad y de seguridad de un producto. No obstante, pienso que no estamos lo suficientemente preparados para evaluar la sostenibilidad, impacto medioambiental, justicia social... En estos aspectos nuestro nivel es meramente suficiente y tenemos que guiarnos más por el sentido común y la intuición que por procedimientos que seguro que existen, pero que desconocemos. En cuanto a herramientas, tanto de proyecto, como de colaboración y trabajo en equipo, creo que también tenemos un buen nivel al acabar la carrera, es el aspecto sostenible/medioambiental/social el que falla.

Sobre la encuesta, tal vez se echa en falta un punto intermedio, una opción neutral. También estaría bien que hubiesen más preguntas en la que se

pudiese desarrollar la respuesta, y que no fuesen todas preguntas con cuatro opciones, aunque entiendo que esto dificulte la comparación entre encuestas respondidas.

En conclusión, la sostenibilidad es algo que aparece en el Grado de manera algo forzada, en competencias transversales, etc. Creo que sería bueno incluirlo de alguna otra manera que no hiciese ver a los alumnos la sostenibilidad como algo que "acabar y quitarse de en medio", si no que se informase y concienciase sobre el tema.

3.5. Leyes y regulaciones

Debido a que se trata de un proyecto que no involucra usuarios, clientes o personas de ningún tipo, y en el cual todo el software utilizado es libre, no hay ninguna ley o regulación que lo afecte.

Lo único a destacar es que el único marco que aplica a los videojuegos es el PEGI, un sistema que clasifica los juegos para determinada edad. *Starcraft 2* está clasificado con PEGI 16, debido a las escenas de violencia y al hecho de disponer de multijugador online.

Capítulo 4

Los escenarios: descripción general

Se han definido un conjunto de 6 escenarios sobre los cuales se entrenarán los agentes. Estos son parte del conjunto de escenarios sobre los cuales DeepMind publicó sus *benchmarks*. Cada uno de estos será detallado en profundidad en su propio apartado, pero se ofrece ahora una visión general sobre ellos.

Se han elegido estos escenarios por diversos motivos. El primero es la posibilidad de comparar nuestros resultados con los mejores benchmarks posibles, los de DeepMind. Sería ingenuo y demasiado optimista superar sus resultados, teniendo en cuenta que DeepMind cuenta con los recursos de Google. Sin embargo, es una buena medida con la que contrastar resultados.

El segundo, es que así disponemos de dos escenarios de movimiento, dos de combate, y dos de *management*. La idea es simular situaciones que puedan darse en una partida normal. Aunque naturalmente, las situaciones reales serían más complejas, los escenarios pueden darnos una idea de cómo son capaces de aprender los agentes estos tres tipos de *gameplay*.

A priori, los agentes deberían comportarse bien en los escenarios de movimiento. Estos son más mecánicos, y es precisamente en mecánicas donde la Inteligencia Artificial debería dar mejores resultados.

Los escenarios de *management* deberían ser los peores. En estos la IA debería ser capaz de desarrollar una estrategia a largo plazo, lo cual es difícil.

En cuanto a los de combate, es difícil de predecir. Aunque involucran una parte mecánica, también es necesario deducir una estrategia de combate apropiada. Por ejemplo, en las competiciones de microcontrol de unidades, los jugadores no solo debían ser buenos manejando las unidades, sino hacerlo de una forma específica, enfrentando sus unidades contra aquellas contra las que eran buenas y viceversa, separándolas para evitar daño de área, etc.

Capítulo 5

Aprendizaje por refuerzo

El aprendizaje por refuerzo es un enfoque de Inteligencia Artificial inspirado por la manera en la que los humanos y los animales aprenden a tomar decisiones a través de recompensas (positivas o negativas) proporcionadas por el entorno. A diferencia de otros enfoques (como el aprendizaje supervisado), en el aprendizaje por refuerzo no contamos con datos o ejemplos de buenos comportamientos o acciones. En cambio, el entrenamiento se hace mediante la interacción del agente con el entorno.

Definamos primero algunos conceptos básicos:

- **Estado (s):** Las características del entorno (y del propio agente) en un instante determinado de tiempo. En nuestro caso, DeepMind define los estados como la información obtenida a partir de los píxeles de la pantalla. En nuestro caso, limitaremos la definición de estados a los parámetros que tenga sentido considerar para cada escenario, reduciendo así la capacidad de computación necesaria.
- **Acción (a):** La maniobra, actividad o movimiento que realiza el agente en un instante determinado de tiempo. Esta acción llevará al agente del estado en que esté **s0** a un estado **s1**, y recibirá una recompensa.
- **Recompensa (r):** La bonificación obtenida por el agente tras realizar una acción.

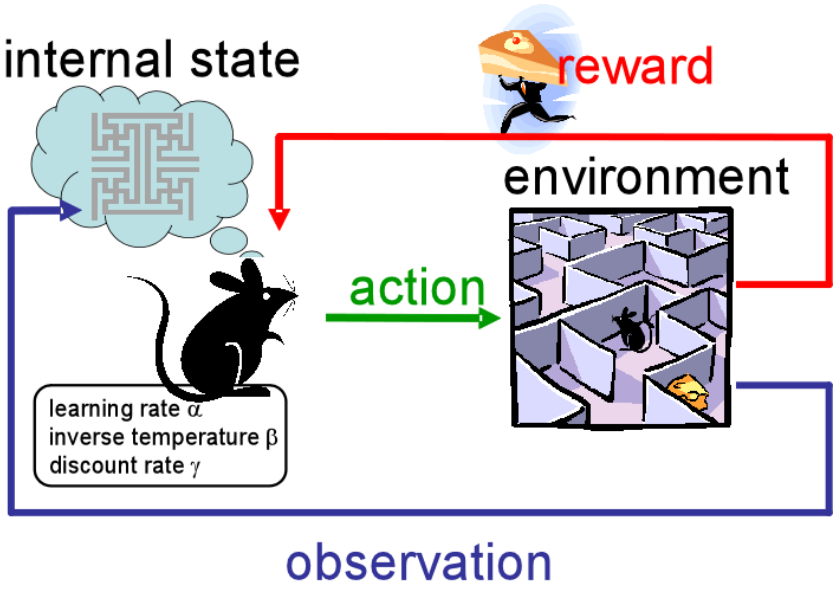

Figura 5.1: Esquema de un típico agente de aprendizaje por refuerzo. *(Imagen extraída de becominghuman.ai)*

El objetivo del agente es descubrir una **política** para seleccionar acciones que maximicen la recompensa a largo plazo. Esta **política** es una estrategia que el agente sigue a la hora de seleccionar acciones a partir del estado en que está. Las interacciones del agente con el entorno ocurren en tiempo discreto (en pasos) y se modelan como un problema de decisión de Markov.

Un problema de decisión de Markov está definido por:

- Un conjunto de estados. Los estados son una función de la información del agente acerca del entorno.
- Un conjunto de acciones posibles en cada estado. Las acciones representan las diferentes maneras en las que un agente puede interactuar con el entorno.
- La probabilidad de transition de un estado **s** a otro estado **s'** dada una acción.
- La función de recompensa en transición de **s** a **s'** dada una acción.

Estas dos últimas definen el modelo del mundo. Si este es conocido, tan solo hay que calcular directamente la estrategia óptima mediante enfoques como la programación dinámica. Si el modelo es, por el contrario, desconocido (como en este caso), hay que aproximar aprendiendo estimaciones de recompensas futuras obtenidas al elegir una acción a en un estado s.

5.1. Conceptos y problemas del aprendizaje por refuerzo

5.1.1. Exploración vs explotación

Una cuestión central en el aprendizaje por refuerzo es el equilibrio entre la **explotación** del conocimiento ya adquirido frente a la **exploración** de nuevos estados y acciones en el espacio de búsqueda. Seleccionar acciones al azar todo el tiempo, o escoger siempre la acción que creemos mejor son ambas opciones pobres.

El mecanismo más popular, que da bastante buenos resultados, se llama ϵ-greedy (epsilon greedy). El parámetro ϵ está entre 0 y 1. El agente que siga esta política, elegirá la opción que crea más favorable con probabilidad 1 - ϵ. Sin embargo, esto por si solo es subóptimo. Supongamos por ejemplo una ϵ de 0.5. El agente elegiría la mejor acción el 50 por ciento de las veces, y el otro 50 por ciento escogería una al azar.

¿Y cuál es el mejor valor? En lugar de dejar uno fijo, se ha optado por un *decay*: empezamos con una ϵ de 1 que se va reduciendo hasta un mínimo de 0.01. La idea es que así, al final el agente elegirá la que considere mejor acción la inmensa mayoría de las veces, pero habrá pasado antes por un amplio espacio de exploración.

5.1.2. Off-policy vs On-policy

A falta de una buena traducción, se utilizará la nomenclatura inglesa. Un agente *off-policy* utiliza aproxima la estrategia óptima independientement de las acciones del agente. Por ejemplo, en Q-Learning, un algoritmo *off-policy* que se detalla en profundidad en capítulos posteriores, se estima el retorno para los pares estado-acción asumiendo que se asume una política *greedy*.

En cambio, en un algoritmo *on-policy* como SARSA (que también se utiliza en este proyecto), el algoritmo aproxima la política como un proceso atado a las acciones del agente, incluyendo los pasos de exploración.

5.1.3. Bootstrapping y Backup

El *bootstrapping* es una noción central dentro del aprendizaje por refuerzo. En pocas palabras, estima como de bueno es un estado basándose en como de bueno **creemos** que es el siguiente. Así, una estimación se actualiza basándose en otra estimación.

Por otro lado, el *backup* es también muy importante, y actúa cono distintivo entre distintos algoritmos de aprendizaje por refuerzo. Mediante el *backup* volvemos desde un estado futuro al actual y consideramos el valor de los estados intermedios en nuestra estimación. Sus propiedades principales son la profundidad y la amplitud.

5.1.4. Tipos de algoritmo

A partir de los conceptos definidos anteriormente encontramos tres tipos distintos de algoritmos de aprendizaje por refuerzo.

1. **Programación dinámica:** En los algoritmos de programación dinámica el conocimiento del model del mundo es necesario y la política óptima es calcula mediante *bootstrapping*.

2. **Métodos de Monte Carlo:** En los métodos de Monte Carlo el conocimiento del model del mundo no es necesario. Los algoritmos de esta clase no utilizan bootstrapping.

3. **TD Learning:** En los métodos de Time Difference Learning (en los que se produce una corrección iterativa de las estimaciones), el modelo de mundo tampoco es requerido. Los algoritmos aprenden mediante *bootstrapping* y variantes de *backup*.

En este proyecto se aplicarán métodos de TD-Learning.

Capítulo 6

Los escenarios: descripción detallada

6.1. Escenario 1: Move to beacon

El primer escenario, **Move to beacon** (o Moverse a baliza), es el más sencillo de todos. Es uno de los dos escenarios centrados en el movimiento de unidades.

6.1.1. Descripción y objetivos

El objetivo del escenario es localizar la baliza y moverse hacia ella. Los agentes deberán alcanzar tantas balizas como sea posible en un tiempo de 2 minutos. Cuando el agente alcanza la baliza, esta se mueve a una localización aleatoria (al menos a 5 unidades de distancia del marine).

Figura 6.1: El agente debe localizar la baliza y moverse hacia ella.

6.1.2. Elementos a tener en cuenta

Es un escenario muy simple. Los únicos elementos a tener en cuenta son el marine que el agente deberá controlar, y la baliza a la que tendrá que llegar.

6.1.3. Recompensa

La recompensa es de +1 cada vez que el marine alcanza la baliza.

6.1.4. Expectativas

Este es un escenario que requiere de poca estrategia, en el que lo más importantes son la velocidad de reacción y la parte mecánica, en la que la IA debería hacer un buen papel.

6.2. Escenario 2: Collect mineral shards

Figura 6.2: Los marines intentan recoger mineral tan rápido como pueden.

El segundo escenario, **Collect mineral shards** (o Recoger fragmentos de mineral), es el segundo de los escenarios de movimiento. Este es bastante más complejo que el primero y requiere cierta estrategia a la hora de mover a las unidades para recoger el mineral.

6.2.1. Descripción y objetivos

El objetivo del escenario es obtener el máximo mineral posible en un período de dos minutos. Si se recogen los 20 minerales del mapa antes de los dos minutos, aparecen 20 más en posiciones aleatorias (al menos a dos unidades de distancia del marine). El escenario consiste de un mapa vacío con minerales esparcidos por el mismo.

6.2.2. Elementos a tener en cuenta

En esta ocasión han de tenerse en cuenta ambas unidades controlables, así como los minerales que se encuentran esparcidos por el mapa. El resto de parámetros del juego no parecen relevantes en este escenario.

6.2.3. Recompensa

La recompensa es de +1 cada vez que un marine recoge un mineral.

6.2.4. Expectativas

Se espera que la IA se comporte peor que en el primero. Esto es debido a que aquí lo importante no es solo moverse a un objetivo, sino que entran en juego factores como los objetivos más cercanos, el camino óptimo, etc. Además, también es relevante la manera de mover las unidades (en grupo, individualmente...). A pesar de esto, se esperan unos resultados aceptables.

6.3. Escenario 3: Defeat roaches

Figura 6.3: El agente intenta eliminar a las cucarachas enemigas.

Este es el primero de los escenarios de combate. El agente deberá aprender no solo a atacar a las cucarachas, sino a hacerlo de una manera determinada si quiere ser eficaz.

6.3.1. Descripción y objetivos

El objetivo de este escenario es eliminar a tantas cucarachas como sea posible en un plazo de 2 minutos. Si eliminan todas las cucarachas, 4 nuevas cucarachas aparecen, así como 5 marines extras de refuerzo con la salud al completo. La salud de los marines supervivientes se mantiene.

6.3.2. Elementos a tener en cuenta

En este escenario hay más parámetros importantes. Para empezar, tenemos las unidades, que no son una o dos sino nueve, y cuyo número va variando a lo largo del escenario. Lo mismo se aplica para las unidades enemigas. Además, hay otros factores en juego como la salud de las unidades, su rango de ataque, la distancia de las unidades enemigas a las aliadas, etc.

6.3.3. Recompensa

La recompensa es de +10 por cada cucaracha eliminada, y de -1 por cada marine perdido.

6.3.4. Expectativas

Aquí es cuando empieza a ser difícil hacer una estimación de los resultados. Dependiendo de la definición del estado, de las acciones y del algoritmo, el agente convergerá sobre una solución u otra.

Teniendo en cuenta los resultados de Deepmind, es posible que alcance o supere el nivel de juego de un jugador estándar. Sin embargo, es poco probable que consiga el control de un Gran Maestro y aprenda estrategias como el "kiting"(alejar momentáneamente a las unidades con poca salud). La estrategia óptima involucraría concentrar el fuego en las cucarachas individualmente.

6.4. Escenario 4: Defeat zerglings and banelings

Figura 6.4: Los marines se enfrentan a las unidades Zerg enemigas.

El segundo de los escenarios de combate es mucho más complejo que el primero. En el anterior, las cucarachas eran una unidad más resistente y poderosa que los marines, que encontraban su punto fuerte en una ventaja numérica. Sin embargo, las mecánicas de ambas unidades eran similares: ambas eran unidades con alcance similar que disparaban a distancia.

En este caso sin embargo, los marines se enfrentan a una combinación diseñada para contrarrestarlos. Si los zerglings (unidades rápidas de ataque méele) alcanzan a los marines, dificultarán su movimiento, facilitando que los banelings impacten contra ellos. Los banelings son unidades suicidas que explotan al contacto, haciendo estragos en un área de efecto. Si impactan contra el grueso de los marines, la derrota está asegurada.

6.4.1. Descripción y objetivos

El objetivo de este escenario es acabar con tantas unidades enemigas como sea posible en un período de dos minutos. Si todas las unidades enemigas son eliminadas antes de los dos minutos, 10 nuevas aparecerán (6 zerglings y 4 banelings), así como 4 marines de refuerzo.

6.4.2. Elementos a tener en cuenta

Además de las unidades aliadas y enemigas, algunos parámetros a considerar son la distancia entre ellas, la salud de cada una de las unidades, si hay banelings vivos, etc.

6.4.3. Recompensa

La recompensa es de +5 por cada zergling o baneling eliminado, y -1 por cada marine perdido.

6.4.4. Expectativas

Parece muy improbable que los agentes desarrollen una estrategia eficaz. Esto implicaría que aprendiesen a identificar el peligro de los banelings, así como a evitarlos. Sería necesario un buen control de las unidades de manera individual, o tal vez “stutter-step”(una maniobra mediante la cual los marines retroceden, disparan, retroceden, disparan...). No parece factible alcanzar el nivel de un jugador humano en este nivel.

6.5. Escenario 5: Collect minerals and gas

Figura 6.5: El agente intenta obtener los máximos recursos posibles.

Este es el primero de los escenarios de gestión. Los escenarios de gestión son los más complejos porque es necesario desarrollar una estrategia a largo plazo, que es punto más débil de los agentes.

6.5.1. Descripción y objetivos

El escenario se compone de un Centro de Mando, donde pueden construirse trabajadores y a donde se llevan los recursos recolectados; dos líneas de minerales que los trabajadores pueden recolectar; 4 géisers de gas vespeno, sobre los que pueden construirse Refinerías para recolectar gas; y doce trabajadores (o VCEs).

El objetivo del escenario es recolectar tantos recursos como sea posible en un plazo de 5 minutos.

6.5.2. Elementos a tener en cuenta

La dificultad de este escenario aumenta también por el número de elementos a tener en cuenta. Algunos de estos son: el Centro de Mando, los VCEs, los minerales, los géisers, las Refinerías (si hay), la población actual, la población total...

6.5.3. Recompensa

La recompensa es igual a la cantidad total de minerales y gas vespeno obtenida.

6.5.4. Expectativas

Es complicado que la IA obtenga una estrategia a largo plazo, especialmente porque las acciones que le ayudarían a conseguir recursos no se verán recompensadas hasta mucho más tarde en el juego, y será difícil que esa recompensa se propague de vuelta y se asocie a esa determinada acción.

Además, la estrategia óptima involucraría la creación de un Centro de Mando adicional, y posiblemente la evolución de estos a Comando Orbital, lo que requiere una visión de futuro que la IA probablemente no pueda obtener.

6.6. Escenario 6: Build marines

Figura 6.6: Es difícil desarrollar una estrategia para construir marines.

El último escenario es el más complejo, ya que se asemeja a un escenario de una partida real. El agente deberá intentar encontrar una estrategia para construir marines. Esto involucrará la creación de edificios como depósitos de suministro y barracones.

6.6.1. Descripción y objetivos

El escenario se compone de un Centro de Mando, donde pueden construirse trabajadores y a donde se llevan los recursos recolectados; una línea de minerales que los trabajadores pueden recolectar; y doce trabajadores (o VCEs).

El objetivo del escenario es recolectar tantos recursos como sea posible en un plazo de 15 minutos.

6.6.2. Elementos a tener en cuenta

La dificultad de este escenario aumenta también por el número de elementos a tener en cuenta. Algunos de estos son: el Centro de Mando, los VCEs, los minerales, los géiseres, las Refinerías (si hay), la población actual, la población total...

6.6.3. Recompensas

La recompensa es igual al número total de marines construidos.

6.6.4. Expectativas

Es complicado que la IA obtenga una estrategia a largo plazo, especialmente porque las acciones que le ayudarían a conseguir recursos no se verán recompensadas hasta mucho más tarde en el juego, y será difícil que esa recompensa se propague de vuelta y se asocie a esa determinada acción.

Capítulo 7

Definición de estados y acciones

Una parte crucial para que los agentes consigan buenos resultados es la definición de estados y acciones. Como se ha mencionado anteriormente, sería inviable para un proyecto de este calibre y características intentar definir un espacio de acciones que incluyese todas las disponibles y un espacio de acciones definido por toda la información en pantalla. Esto no solo resultaría en tiempos de entrenamiento muy superiores al total de horas establecido para el proyecto, sino que dificultaría muchísimo el aprendizaje.

Para solventar este problema, este proyecto llevará a cabo su propia definición del espacio de acciones y estados para cada escenario. Naturalmente, esto afectará a la capacidad de aprendizaje de los agentes por lo que hay que ser cuidadoso para no pasar por alto ningún factor importante.

7.1. Definición de estados

Los estados se definirán a partir de los parámetros que tengan sentido y sean relevantes para cada escenario. Un buen punto de partida son los elementos a tener en cuenta para cada escenario, descritos previamente. De esta manera se reducirá el espacio de exploración.

7.2. Definición de acciones

Se limitarán las acciones que no sean relevantes para el escenario. Así se reducirá mucho el espacio de acciones, eliminando todas aquellas no útiles (que la IA no habría aprendido igualmente) y facilitando el entrenamiento. Destacar que incluso DeepMind llevó a cabo este mismo proceso para el último escenario debido a su complejidad.

Sin embargo, a pesar de reducir las acciones disponibles, el aprendizaje es aún muy complicado. Para facilitarlo, se introducirán las acciones con semántica.

7.2.1. Acciones con semántica

DeepMind ya redujo el espacio de acciones con su API, pero este sigue siendo extremadamente grande. En *Starcraft 2*, sin embargo, pueden observarse secuencias de acciones cuyo único propósito es combinarse para llevar a cabo una acción de más alto nivel.

Por ejemplo, supongamos que queremos construir un barracón. En lugar de hacer acciones de nivel tan bajo como hacer click en una coordenada determinada, DeepMind ya estableció la acción "build_barracks", que presiona el botón de construir barracón en unas coordenadas determinadas.

Esto puede llevarse aún a más alto nivel, y es que para construir un barracón, primero debemos haber seleccionado un trabajador, lo que implicaría identificar de algún modo las coordenadas de un trabajador, ejecutar la acción "select_point"sobre ese punto, y después ejecutar "build_barracks".

Este conjunto de acciones se lleva a cabo todo el tiempo. Cuando un jugador humano piensa en construir un edificio, realiza esta secuencia de acciones que tienen como objetivo realizar la acción global. Así podemos empaquetar estas secuencias para reducir el número de acciones, hacerlas más lógicas y facilitar el aprendizaje. En este caso, construiríamos nuestra propia acción "_build_barracks", que constaría de dos pasos: el primero en el que seleccionaríamos un trabajador y el segundo en el que ordenaríamos la

construcción del barracón.

Podemos utilizar esta técnica para construir acciones más complejas, utilizando ingeniería del conocimiento y si bien es cierto que se está introduciendo conocimiento externo en el sistema, es la solución propuesta para facilitar el aprendizaje, para compensar el déficit en recursos computacionales.

Capítulo 8

Q-Learning

8.1. Descripción

Q-Learning[13][14][15][12] es un algoritmo de TD-Learning, *off-policy*, que se basa en una representación tabular de valores Q(s,a). Informalmente, Q(s,a) representa como de bueno es elegir la acción a en el estado s. Más formalmente, Q(s,a) es el refuerzo descontado esperado de realizar la acción a en el estado s. Q-Learning aprende eligiendo acciones y recibiendo recompensas mediante *bootstrapping*.

8.2. ¿Cómo funciona?

El objetivo de Q-Learning es maximizar la recompensa esperada eligiendo la acción óptima en cada estado. El algoritmo no es complicado. Consiste en una actualización de los *Q values* de forma iterativa.

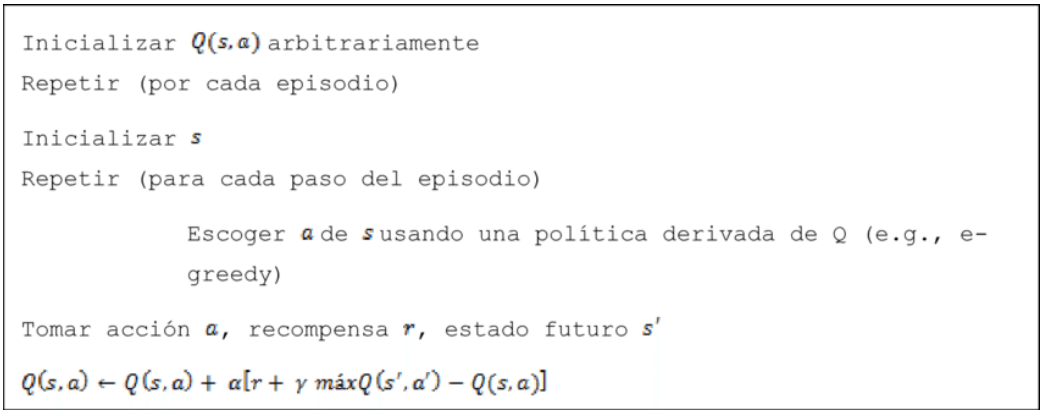

Figura 8.1: Pseudocódigo para Q-Learning.

Inicialmente, la tabla se inicializa con valores arbitrarios. Cada vez que el agente selecciona una acción a de un estado s, va a un estado *s'*, recibe una recompensa inmediata r y actualiza su valor Q(s,a) de la siguiente manera:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\{r + \lambda maxQ(s',a') - Q(s,a)\}$$

$\alpha \in [0,1]$ es el **ratio de aprendizaje** y $\lambda \in [0,1]$ es el **factor de descuento**. El ratio de aprendizaje determina hasta que punto la nueva estimación de Q sobrescribirá la anterior. El factor de descuento pondera la importancia de las recompensas recientes frente a los posteriores: cuanto más cercano a 1, mayor es el peso de los futuros refuerzos.

Como puede verse en la ecuación, el algoritmo utiliza *bootstrapping* ya que mantiene estimadores de como de bueno es un par acción-estado basando en como de bueno cree que es el siguiente estado.

Además utiliza un *backup* de 1 paso de profundidad y amplitud completa para estimar Q, tomando en consideración todos los *Q values* de las posibles acciones *a'* de un nuevo estado *s'*.

Capítulo 9

Agentes de Q-Learning

Para todos estos agentes, a no ser que se especifique lo contrario, se asume un ratio de aprendizaje de 0.01 y un factor de descuento de 0.9 (elevado, para dar más importancia a las recompensas a largo plazo).

9.1. Move to beacon

9.1.1. Definición de estado

El estado en este escenario es bastante simple, ya que hay muy pocos elementos a tener en cuenta.

El estado se define por los siguientes elementos:

- **marine_selected**: Determina si el marine está seleccionado o no.
- **marine_on_beacon**: Determina si el marine está sobre la baliza o no.
- **beacon_pos**: Las coordenadas de la baliza.
- **marine_coords**: Las coordenadas del marine.

9.1.2. Definición de acciones

En este escenario las acciones pueden limitarse mucho. Las escogidas son las siguientes.

- **_NO_OP:** No realizar ninguna acción. Esta acción tiene sentido cuando el agente quiere permanecer estacionario, o cuando no hay ninguna otra acción disponible. Estará presente en todos los agentes.
- **_SELECT_ARMY:** Seleccionar todas las unidades del ejército (en este caso el marine).
- **_SELECT_POINT:** Selecciona un punto del mapa.
- **_MOVE_RAND:** Mueve el marine a una posición al azar (si el marine está seleccionado).
- **_MOVE_MIDDLE:** Mueve el marine al centro del mapa (si el marine está seleccionado).
- **_MOVE_TO_BEACON:** Mueve el marine a las coordenadas de la baliza.

En este caso no tiene sentido una acción "_SELECT_MARINE", ya que al haber uno solo, la acción de seleccionar el ejército cumple la misma función sin requerir coordenadas.

9.1.3. Entrenamiento y resultados

Tras entrenar el agente, este aprendió que la mejor opción era seleccionar el marine cuando este no estaba ya seleccionado, y cuando si lo estaba moverse hacia la baliza todo el tiempo.

Puntuación media: 26

Puntuación máxima: 32

Si comparamos los resultados con los de DeepMind vemos que está a la altura de los 3 mejores agentes, así como de un Gran Maestro. Esto cumple con las expectativas, ya que es un escenario muy mecánico y bastante simple.

9.2. Collect mineral shards

9.2.1. Definición de estado

De nuevo nos encontramos con un escenario bastante sencillo en cuanto a estado. La dificultad se encuentra en la multiplicidad: en lugar de una unidad controlable tenemos dos, y en lugar de un objetivo al que ir, hay decenas.

El estado se define por los siguientes elementos:

- **marine_selected**: Determina si alguno de los marines está seleccionado.
- **army_selected**: Determina si ambos marines están seleccionados. Los resultados de las acciones de movimiento variarán en función de si hay una única unidad seleccionada o las dos (o ninguna). Por ello es importante tenerlo en cuenta en el estado.

En este caso no se guardarán las coordenadas de los minerales por varios motivos. Tras haber llevado a cabo el entrenamiento del primer agente, resultó aparente que las coordenadas no tenían impacto. Esto tiene sentido, ya que el objetivo del agente, independientemente de las coordenadas del objetivo, es siempre el mismo: moverse hacia él.

Además, almacenar las coordenadas como parte del estado genera un número muy elevados de estados distintos que en realidad son, a todos los efectos, el mismo.

Se realizaron pruebas con otra definición del estado que incluía:

- **last_action**: La última acción que ha realizado el agente.

La idea era ver si teniendo en cuenta la última acción que había realizado, el agente era capaz de aprender patrones. Sin embargo los resultados fueron idénticos.

9.2.2. Definición de acciones

En este escenario definimos un espacio de acciones mayor. Las escogidas son las siguientes.

- **_NO_OP:** No realizar ninguna acción. Esta acción tiene sentido cuando el agente quiere permanecer estacionario, o cuando no hay ninguna otra acción disponible. Estará presente en todos los agentes.
- **_SELECT_ARMY:** Seleccionar todas las unidades del ejército.
- **_SELECT_POINT:** Selecciona un punto del mapa.
- **_SELECT_MARINE:** Selecciona un marine.
- **_MOVE_RAND:** Mueve el marine a una posición al azar (si el marine está seleccionado).
- **_MOVE_MIDDLE:** Mueve el marine al centro del mapa (si el marine está seleccionado).
- **_MOVE_TO_RANDOM_MINERAL:** Mueve las unidades seleccionadas a un mineral aleatorio.
- **_MOVE_TO_CLOSEST_MINERAL:** Mueve las unidades seleccionadas al mineral más cercano.

Además, se incluyen acciones semánticas de dos o más pasos:

- **_MARINE_TO_CLOSEST_MINERAL:** Mueve un marine a su mineral más cercano. Consta de dos pasos, seleccionar el marine y moverlo.

- **_MARINE_TO_RANDOM_MINERAL:** Mueve un marine a un mineral aleatorio. Consta de dos pasos, seleccionar el marine y moverlo.
- **_ARMY_TO_CLOSEST_MINERAL:** Mueve a ambos marines a sus minerales más cercano. Consta de cuatro pasos, seleccionar cada uno de los marines y moverlos.
- **_ARMY_TO_CLOSEST_MINERAL:** Mueve a ambos marines a un mineral aleatorio. Consta de cuatro pasos, seleccionar cada uno de los marines y moverlos.

Las acciones no semánticas pueden parecer débiles a primera vista frente a las semánticas. Esto no tiene porque ser así. Las acciones semánticas son extras que sirven para guiar al agente a partir de patrones conocidos basados en el conocimiento del juego. Sin embargo, la IA podría desarrollar sus propios patrones: podría hacer su propio sistema de selección con **_SELECT_POINT**, podría descubrir que es más óptimo mover a ambos marines al punto central del mapa y distribuirlos desde ahí, o tal vez moverlos a puntos aleatorios favorecerá la recolección, debido a la alta densidad de minerales en el mapa.

El objetivo es limitar las acciones a aquellas que sean lógicas para facilitar el entrenamiento, pero dejando margen para que los agentes encuentren sus propias estrategias (si es que las hay más óptimas).

9.2.3. Entrenamiento y resultados

El agente aprendió aprendió a recolectar los minerales de manera eficiente, controlando las unidades por separado.

Puntuación media: 92

Puntuación máxima: 107

Aunque se queda por debajo de las puntuaciones de un jugador de nivel alto como el de DeepMind, y más aún de un GrandMaster, el agente aprende

a resolver el escenario satisfactoriamente, con un puntuación ligeramente inferior a la de los dos mejores agentes de DeepMind.

Probablemente la diferencia entre un jugador y el agente es la posibilidad de planear el futuro. Seguramente un jugador humano, al ver la distribución de los minerales, determinaría un camino óptimo o casi para cada uno de los marines que minimizase el tiempo de recolección. Sin embargo, el agente no ha sido capaz.

Cabe destacar que aunque a priori las acciones semánticas de 4 pasos parecían más óptimas, el agente acabó aplicando con más frecuencia las de dos pasos. Tras un análisis, esto es debido a que las acciones de 4 pasos, que aplicaban política Round Robin para seleccionar los marines (seleccionar marine 1, moverlo, seleccionar marine 2, moverlo, repetir), causaban que durante breves instantes los marines permaneciesen estáticos. La causa es que si al mover a uno de los dos marines, este alcanzaba un objetivo, durante los dos siguientes pasos (actuación del otro marine) no se movía al siguiente objetivo.

De hecho, se realizó una prueba eliminando las acciones semánticas de dos pasos (de manera que el agente utilizó con más frecuencia las de 4) y el agente solo consiguió puntuaciones de:

Puntuación media: 79

Puntuación máxima: 95

Así, el agente detectó esté problema y pudo adaptarse al mismo.

9.3. Defeat roaches

9.3.1. Definición de estado

Nos encontramos en el primer escenario centrado en el combate. Según la dificultad va subiendo, los estados y acciones incrementan su complejidad. Aún así, la complejidad de esta definición de estado no es muy elevada.

El estado se define por los siguientes elementos:

- **army_selected:** Determina si el ejército está seleccionado.
- **more_allies:** Determina si hay más unidades aliadas que enemigas.
- **unit_in_danger:** Determina si hay alguna unidad en peligro.
- **unit_in_danger_selected:** Determina si hay alguna unidad en peligro seleccionada.
- **last_action:** La última acción realizada por el agente.

De nuevo, la clave es evitar parámetros continuos y usar otros booleanos que representen los aspectos más importantes. Por ejemplo, la salud de las unidades como parte del estado generaría un número prohibitivo de los mismos, así que mientras sea posible, se evitará este tipo de parámetros.

El parámetro **more_allies** podría servir para que el agente tome estrategias más conservadores cuando esté en inferioridad de condiciones, evitando que su puntuación disminuya al perder marines.

Los parámetros referentes a las unidades en peligro están pensados para dar la posibilidad al agente de modificar su juego cuando haya unidades con poca salud cerca de los enemigos.

Sin embargo la definición de "en peligro" es un tanto ambigua. ¿Está una unidad en peligro cuando su salud se encuentra por debajo del 50 %? ¿O del 25 %? ¿Y si está muy cerca de las unidades enemigas?

Se experimentará con distintas opciones definiciones de "en peligro".

9.3.2. Definición de acciones

Las escogidas son las siguientes.

- **_NO_OP:** No realizar ninguna acción. Esta acción tiene sentido cuando el agente quiere permanecer estacionario, o cuando no hay ninguna otra acción disponible. Estará presente en todos los agentes.
- **_SELECT_ARMY:** Seleccionar todas las unidades del ejército.
- **_SELECT_POINT:** Selecciona un punto del mapa.
- **_SELECT_MARINE:** Selecciona un marine.
- **_MOVE_RAND:** Mueve el marine a una posición al azar (si el marine está seleccionado).
- **_MOVE_MIDDLE:** Mueve el marine al centro del mapa (si el marine está seleccionado).
- **_MOVE_TO_RANDOM_TARGET:** Mueve las unidades seleccionadas a un objetivo aleatorio.
- **_MOVE_TO_CLOSEST_TARGET:** Mueve las unidades seleccionadas al objetivo más cercano.
- **_ATTACK_RANDOM_TARGET:** Ataca a un enemigo aleatorio.
- **_ATTACK_CLOSEST_TARGET:** Ataca al enemigo más cercano.
- **_ATTACK_LOWEST_HP:** Ataca al enemigo con menos salud.
- **_KITE_ARMY:** Mueve el ejército una pequeña distancia en dirección opuesta al grueso de las fuerzas enemigas.
- **_KITE_UNIT:** Mueve una unidad una pequeña distancia en dirección opuesta al grueso de las fuerzas enemigas.
- **_KITE_LOWEST_HP:** Mueve la unidad aliada con menos salud una pequeña distancia en dirección opuesta al grueso de las fuerzas enemigas.
- **_SELECT_DANGER_UNIT:** Selecciona una unidad en peligro.

Además de las acciones de selección y movimiento, se añaden distintas acciones de ataque. Además, las últimas 4 acciones proporcionan al agente opciones para retirar unidades del combate, si es que considera que es la mejor opción.

9.3.3. Entrenamiento y resultados

El agente aprende una estrategia de combate efectiva, consiguiendo superar por bastante los resultados del jugador medio. Sin embargo, no es capaz de aprender técnicas más avanzadas, situándose por debajo del Gran Maestro.

Puntuación media: 98

Puntuación máxima: 336

El agente, a la altura de los de DeepMind, desarrolla una estrategia casi óptima basándose en concentrar el fuego en la misma unidad enemiga, en lugar de repartirlo entre todas. Aunque el daño por segundo global sea el mismo, concentrar el fuego sirve para eliminar antes a cada unidad, de manera que se reduce el daño total recibido, ya que hay menos enemigos atacando.

Sin embargo, no ha sido capaz de igualarse a la estrategia de un Gran Maestro. Esto es seguramente debido a que el agente no aprende a retirar las unidades cuando están a punto de morir.

9.4. Defeat zerglings and banelings

9.4.1. Definición de estado

El segundo escenario de combate es mucho más difícil y por tanto ese necesitara una definición de estado más compleja para ofrecer la posibilidad al agente de refinar su estrategia.

El estado se define por los siguientes elementos:

- **army_selected:** Determina si el ejército está seleccionado.
- **more_allies:** Determina si hay más unidades aliadas que enemigas.
- **unit_in_danger:** Determina si hay alguna unidad en peligro.

- **unit_in_danger_selected:** Determina si hay alguna unidad en peligro seleccionada.
- **last_action:** La última acción realizada por el agente.
- **enemies_in_sight:** Determina si hay enemigos a la vista.
- **zerglings_in_sight:** Determina si hay zerglings a la vista. En caso contrario, significa que todos los zerglings han sido eliminados.
- **baneling_in_sight:** Determina si hay banelings a la vista. En caso contrario, signifca que todos los banelings han sido eliminados.

Las últimas tres acciones se añaden con el propósito de que el agente sea capaz de aprender distintas tácticas en función de los enemigos que tenga enfrente. Por ejemplo, tendría sentido que fuese más conservador cuando hay banelings en juego, pero que fuese más ofensivo cuando estos ya han sido eliminados.

Se experimentó con un parámetro extra **_ALREADY_SPLIT**, que indicaba si el ejército ya había sido separado, para ver si el agente modificaba su juego, pero los resultados fueron los mismos.

9.4.2. Definición de acciones

Las escogidas son las siguientes.

- **_NO_OP:** No realizar ninguna acción. Esta acción tiene sentido cuando el agente quiere permanecer estacionario, o cuando no hay ninguna otra acción disponible. Estará presente en todos los agentes.
- **_SELECT_ARMY:** Seleccionar todas las unidades del ejército.
- **_SELECT_MARINE:** Selecciona un marine.
- **_MOVE_RAND:** Mueve el marine a una posición al azar (si el marine está seleccionado).

- **_MOVE_MIDDLE:** Mueve el marine al centro del mapa (si el marine está seleccionado).
- **_MOVE_TO_RANDOM_TARGET:** Mueve las unidades seleccionadas a un objetivo aleatorio.
- **_MOVE_TO_CLOSEST_TARGET:** Mueve las unidades seleccionadas al objetivo más cercano.
- **_ATTACK_RANDOM_TARGET:** Ataca a un enemigo aleatorio.
- **_ATTACK_CLOSEST_TARGET:** Ataca al enemigo más cercano.
- **_ATTACK_LOWEST_HP:** Ataca al enemigo con menos salud.
- **_ATTACK_CLOSEST_BANELING:** Ataca al baneling más cercano.
- **_ATTACK_CLOSEST_ZERGLING:** Ataca al zergling más cercano.
- **_ATTACK_LOWEST_HP_BANELING:** Ataca al baneling con menos salud.
- **_ATTACK_LOWEST_HP_ZERGLING:** Ataca al zergling con menos salud.
- **_SPLIT_ARMY:** Separa las unidades.
- **_KITE_ARMY:** Mueve el ejército una pequeña distancia en dirección opuesta al grueso de las fuerzas enemigas.
- **_KITE_UNIT:** Mueve una unidad una pequeña distancia en dirección opuesta al grueso de las fuerzas enemigas.
- **_KITE_LOWEST_HP:** Mueve la unidad aliada con menos salud una pequeña distancia en dirección opuesta al grueso de las fuerzas enemigas.
- **_SELECT_DANGER_UNIT:** Selecciona una unidad en peligro.

Se le han proporcionado al agente acciones extras de ataque para los distintos tipos de unidades enemigas, además de una acción para dispersar sus unidades.

9.4.3. Entrenamiento y resultados

El agente se queda muy lejos de los resultados de un jugador humano (aunque iguala a los agentes de DeepMind). La estrategia que desarrolla se base en separar continuamente sus unidades. Separarlas es muy útil, ya que evita el daño de área de los banelings. Sin embargo, una vez están separadas, hacer *stutter-step*, huir con las unidades heridas, o concentrar fuego en los banelings lograrían mejores resultados.

Puntuación media: 83

Puntuación máxima: 196

Este es un escenario difícil de aprender, que requiere tener muchos factores en cuenta. Sin embargo, una representación más compleja del estado no tiene porque garantizar resultados (como pone de manifiesto DeepMind).

Si comparamos los resultados con los *benchmarks* actuales son decentes, pero si los comparamos con el nivel humano, este agente puede considerarse el primer fracaso.

Destacar que la media es equiparable a la de DeepMind, con máximos muy inferiores, lo que sugiere que el agente consigue resultados algo más elevados en general.

9.5. Collect minerals and gas

9.5.1. Definición de estado

Siendo este uno de los dos escenarios de gestión, el agente requerirá información más detallada sobre aspectos que no se habían tenido en cuenta antes.

El estado se define por los siguientes elementos:

- **supply_block_risk:** Determina si al agente le queda poco para quedarse bloqueado de suministros (impide construir más unidades).
- **supply_block:** Determina si el agente se encuentra en bloqueo de suministros.
- **orbital_command:** Determina la existencia de algún Comando Orbital.
- **depot_built:** Determina la existencia de algún depósito de suministros.
- **barracks_built:** Determina la existencia de algún barracón.
- **refineries:** Determina el número de refinerías construidas. Se trata de una variable teóricamente continua, pero que no incrementa exageradamente el número de estados ya que por limitación del mapa, como máximo puede haber 4..
- **CC_selected:** Determina si el Centro de Mando está seleccionado.
- **idle_workers:** Determina si hay trabajadores inactivos.

Esta información debería ser suficiente para el agente. Los parámetros **depot_built** y **barracks_built**, a priori no muy relevantes, se incluyen por un motivo específico. Existe una unidad llamada Mula con la velocidad de recolección de 6 trabajadores. Esta se invoca desde el Comando Orbital, que para hacerse requiere un Barracón, que a su vez requiere un Depósito de Suministros.

Así, se le proporciona al agente una manera de llegar a esa estrategia.

9.5.2. Definición de acciones

Las escogidas son las siguientes.

- **_NO_OP:** No realizar ninguna acción. Esta acción tiene sentido cuando el agente quiere permanecer estacionario, o cuando no hay ninguna otra acción disponible. Estará presente en todos los agentes.

- **_SELECT_CC:** Selecciona el Centro de Mando.
- **_SELECT_SCV:** Selecciona un trabajador.
- **_SELECT_IDLE_WORKER:** Selecciona un trabajador inactivo.
- **_SELECT_IDLE_WORKERS:** Selecciona todos los trabajadores inactivos.
- **_IDLE_WORKER_TO_MINERAL:** Manda a un trabajador inactivo a una veta de mineral.
- **_IDLE_WORKERS_TO_MINERAL:** Manda a los trabajadores inactivos a una veta de mineral.
- **_RALLY_WORKERS_MINERAL:** Coloca el punto de reunión del Centro de Mando (donde van los trabajadores tras ser creados) en una veta de mineral.
- **_MOVE_TO_MINERAL:** Manda a la unidad o unidades seleccionadas a una veta de mineral.
- **_MOVE_TO_VESPENE:** Manda a la unidad o unidades seleccionadas a una refinería de gas vespeno (si es que existe alguna).
- **_BUILD_SUPPLY_DEPOT:** Construye un depósito de suministros.
- **_BUILD_SCV:** Construye un trabajador.
- **_BUILD_REFINERY:** Construye una Refinería.
- **_BUILD_BARRACKS:** Construye un Barracón.
- **_BUILD_CC:** Construye un Centro de Mando.
- **_MORPH_ORBITAL:** Convierte un Centro de Mando en un Comando Orbital.
- **_CALL_MULE:** Invoca una Mula.

Con este conjunto de acciones y estados, el agente tiene las herramientas necesarias para aprender el escenario.

9.5.3. Entrenamiento y resultados

A pesar de varios intentos y algunas modificaciones (como aumentar el factor de descuento a 0.99 para dar aún más peso a las recompensas futuras), el agente no es capaz de aprender el escenario en absoluto. El mejor resultado obtenido es que el agente deduzca que es positivo enviar a los trabajadores al mineral (que lo es), y empiece a moverlos todo el rato de un mineral a otro sin dejarles recolectar nada.

Puntuación media: El agente fracasa.

Puntuación máxima: El agente fracasa.

Hay diversas causas posibles para este fracaso:

1. El espacio de estados y acciones es demasiado grande para una representación tabular.
2. El algoritmo no es lo bastante potente.
3. La asociación de recompensas con acciones es demasiado difícil, y se producen demasiados falsos positivos (se explicará en la sección 12.3 del capítulo 12).
4. El problema es sencillamente demasiado complejo.

9.6. Build Marines

9.6.1. Definición de estado

El segundo escenario de gestión es aún más complejo que el primero, ya que es prácticamente una extensión del mismo.

El estado se define por los mismos que el agente anterior excepto por:

- Se elimina el parámetro **refineries**, ya que no hay géiseres de vespeno en este mapa.

Si el agente anterior no fue capaz de aprender, las expectativas para este son muy bajas. Sin embargo, la definición no parece la principal causa. Aún así, simplificarlo con la ausencia de refinerías puede que facilite el aprendizaje.

9.6.2. Definición de acciones

Las escogidas son las mismas que en el agente anterior, salvo por:

- Se eliminan las acciones **_MOVE_TO_VESPENE** y **_BUILD_REFINERY**, ya que no hay gas vespeno en este mapa.
- Se elimina la acción **_BUILD_CC**, ya que no hay una segunda línea de minerales en este mapa.
- Se añade la acción **_BUILD_MARINE**, que permite construir un marine.

9.6.3. Entrenamiento y resultados

El agente, al igual que el anterior, falla a la hora de desarrollar una estrategia para cumplir con su objetivo. Las posibles causas de este fracaso son las mismas que en escenario anterior.

Puntuación media: El agente fracasa.

Puntuación máxima: El agente fracasa.

Lo único bueno a extraer de esto es que los resultados de DeepMind también son nulos. Sus agentes no son capaces de aprender este último escenario, lo que resalta aún más la dificultad del mismo.

9.7. Conclusiones

Los agentes de Q-Learning han hecho en los cuatro primeros escenarios, igualando a los de DeepMind y superando el nivel humano en algunos casos.

AGENT	METRIC	MOVETOBEACON	COLLECTMINERALSHARDS	DEFEATROACHES	DEFEATZERGLINGSANDBANELINGS	COLLECTMINERALSANDGAS	BUILDMARINES
Random Policy	MEAN	1	17	1	23	12	<1
	MAX	6	35	46	118	750	5
Random Search	MEAN	25	32	51	55	2318	8
	MAX	29	57	241	159	3940	46
DeepMind Human Player	MEAN	26	133	41	729	6880	138
	MAX	28	142	81	757	6952	142
Starcraft GrandMaster	MEAN	28	177	215	727	7566	133
	MAX	28	179	363	848	7566	133
Atari-Net	BEST MEAN	25	96	101	81	3356	<1
	MAX	33	131	351	352	3505	20
FullyConv	BEST MEAN	26	103	100	62	3978	3
	MAX	45	134	355	251	4130	42
FullyConv LSTM	BEST MEAN	26	104	98	96	3351	6
	MAX	35	137	373	444	3995	62
Q-Learning	MEAN	26	92	98	83	<1	<1
	MAX	32	107	336	196	<1	<1

Figura 9.1: Tabla de resultados incluyendo Q-Learning

La reducción del espacio de acciones y estados ha equilibrado la balanza y compensado la falta de recursos computacionales.

Lamentablemente, los agentes ni se han acercado a dominar los escenarios de gestión. Es difícil explicar el motivo exacto en este punto, así que se harán pruebas con otros dos algoritmos, SARSA y DQN. Para estos agentes, la definición de agentes y estados será la misma a no ser que se indique específicamente lo contrario.

Capítulo 10

SARSA

10.1. Descripción

SARSA[16][17][15] es un algoritmo de TD-Learning, muy similar a Q-Learning. La diferencia principal, es que SARSA es *on-policy*, en lugar de *off-policy*. SARSA tiene en cuenta la política que el agente esta siguiendo para realizar sus estimaciones, mientras que Q-Learning asume que se sigue una política óptima.

10.2. ¿Cómo funciona?

El nombre SARSA no es más que las siglas correspondientes a State-Action-Reward-State-Action (Estado-Acción-Recompensa-Estado-Acción), que describen muy bien el comportamiento de SARSA.

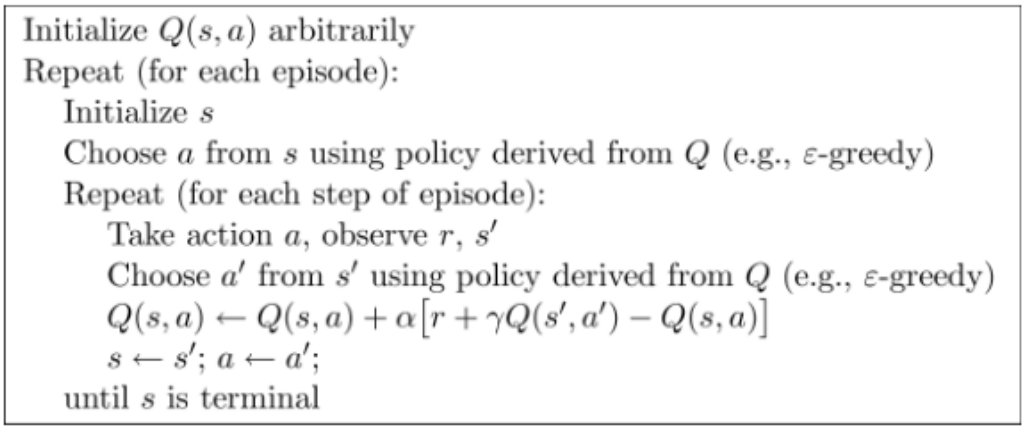

Figura 10.1: Pseudocódigo de SARSA

De hecho la única diferencia con Q-Learning es que donde Q-Learning escoge la mejor acción de un estado s', SARSA elige una acción a' siguiendo la política que esté utilizando el agente.

Destacar que aunque lo más común es utilizar solo un paso, pueden usarse más. Sin embargo, esto implica guardar la tabla de probabilidad de todas las secuencias lo que aumenta exponencialmente el espacio necesario. Incluso si se usasen Redes Neuronales, la complejidad del modelo aumentaría de manera exponencial.

Capítulo 11

Agentes de SARSA

11.1. Move to beacon

11.1.1. Definición del estado

El estado se define a partir de las coordenadas de la baliza y las de el marine, así como de si este está seleccionado o no. No se han hecho modificaciones respecto a Q-Learning.

11.1.2. Definición de las acciones

Se incluyen acciones distintas de movimiento, así como la opción de **_NO_OP**. No se han hecho modificaciones respecto a Q-Learning.

11.1.3. Entrenamiento y resultados

Puntuación media: 26

Puntuación máxima: 32

Los resultados del primer escenario son idénticos con SARSA a Q-Learning. Esto era de esperar, debido tanto al hecho de que el escenario es simple, como al hecho de que Q-Learning ya había obtenido resultados óptimos en este escenario, así que SARSA como mucho habría empeorado.

11.2. Collect Mineral Shards

11.2.1. Definición del estado

Se define el estado a partir de si el ejército o algún marine está seleccionado, y se realizaron pruebas añadiendo la última acción. No se han hecho modificaciones respecto a Q-Learning.

11.2.2. Definición de las acciones

Se definen diversas acciones de selección y movimiento, así como acciones semánticas de dos o cuatro pasos con comportamientos algo más complejos. No se han hecho modificaciones respecto a Q-Learning.

11.2.3. Entrenamiento y resultados

En este caso, SARSA no mejora los resultados previos. De nuevo, esto es de esperar, ya que para los primeros escenarios, Q-Learning ya ha conseguido resultados muy buenos. La incertidumbre yace más en los escenarios de gestión.

SARSA consigue máximos más altos, pero esto se debe al factor aleatorio de la distribución de minerales en el mapa, y no a una mejor estrategia.

Puntuación media: 93

Puntuación máxima: 117

Cabe destacar que la misma prueba que se hizo con Q-Learning se llevó a cabo con SARSA, eliminando las acciones semánticas de dos pasos pero dejando las de cuatro. En este caso, SARSA si se diferenció de Q-Learning, obteniendo puntuaciones significativamente mejores de:

Puntuación media: 64

Puntuación máxima: 79

Esto puede deberse al hecho de que SARSA es más conservador y evita más riesgos que Q-Learning, ya que SARSA intenta converger teniendo en cuenta las posibles penalizaciones de los movimientos de exploración, como se puede ver en detalle en el problema del Cliff Walking[12].

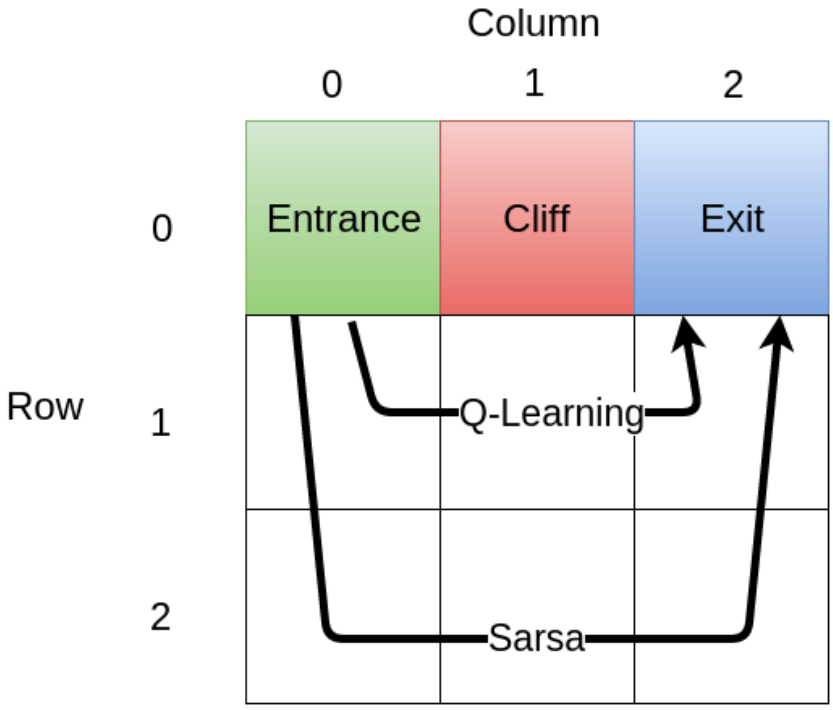

Figura 11.1: En el problema de Cliff Walking, Q-Learning aprende una estrategia a distinta a SARSA. La de Q-Learning es más óptima en cuanto a distancia, pero también más arriesgada. SARSA en cambio opta por una estrategia más segura. *(Imagen extraída de stackexchange.com)*

Por tanto, es lógico que los resultados sean similares o ligeramente a favor de Q-Learning.

11.3. Defeat Roaches

11.3.1. Definición del estado

El estado se define respecto a si el ejército está seleccionado o no, la última acción, las unidades en peligro y el número de aliados respecto al de enemigos. No se han hecho modificaciones respecto a Q-Learning.

11.3.2. Definición de las acciones

Se definen distintas acciones de movimiento, selección y combate. No se han hecho modificaciones respecto a Q-Learning.

11.3.3. Entrenamiento y resultados

SARSA aprende la misma política que Q-Learning, con resultados prácticamente idénticos. La diferencia en la media es debido a la aleatoriedad del mapa.

Puntuación media: 101

Puntuación máxima: 336

11.4. Defeat Zerglings and Banelings

11.4.1. Definición del estado

El estado se define respecto a si el ejército está seleccionado o no, la última acción, las unidades en peligro, el número de aliados respecto al de enemigos, los tipos de unidades enemigas y si el ejército está separado o no. No se han hecho modificaciones respecto a Q-Learning.

11.4.2. Definición de las acciones

Se definen distintas acciones de movimiento, selección y combate. No se han hecho modificaciones respecto a Q-Learning.

11.4.3. Entrenamiento y resultados

De nuevo los resultados han sido parecidos a los de Q-Learning. A priori SARSA podría haber aprendido una estrategia más conservadora, que le otorgase menos puntos pero recibiese menos penalización por perder marines. Sin embargo, al final ha desarrollado una estrategia como la de Q-Learning.

Puntuación media: 82

Puntuación máxima: 192

11.5. Collect Minerals and Gas y Build Marines

11.5.1. Definición del estado

Se ha definido el estado basándose en los edificios contruidos, en los límites de población y en el estado de los trabajadores. No se han hecho modificaciones respecto a Q-Learning.

11.5.2. Definición de las acciones

Se han definido acciones de selección y movimiento de trabajadores y construcción y mejora de edificios. No se han hecho modificaciones respecto a Q-Learning.

11.5.3. Entrenamiento y resultados

Puntuación media: El agente fracasa.

Puntuación máxima: El agente fracasa.

Tal y como era de esperar, los resultados de SARSA es estos dos escenarios son igual de pobres que con Q-Learning. Los agentes no logran aprender ninguna estrategia viable que logre resolver el problema.

11.6. Conclusiones

Las diferencias entre Q-Learning y SARSA son mínimas. Lo más destacable es la diferencia de puntuaciones en la prueba adicional del escenario **Collect Mineral Shards**, debido al carácter más conservador de SARSA.

Si bien es cierto que para la complejidad de *Starcraft 2*, un solo paso en SARSA tal vez no sea suficiente, no se cree que esta sea la raíz del problema, ni el camino para solucionarlo. Primero, porque habría que utilizar un número elevado de pasos (ni uno ni dos), lo que incrementaría el coste espacial exageradamente. Y segundo, porque la hipótesis en este punto es que las causas principales del fracaso son dos, que están relacionadas:

- La naturaleza tabular hace no solo que sea difícil predecir para estados que se dan con menos frecuencia, sino que también hace más difícil que los efectos de una acción se propaguen.
- Existe un gran número de falsos positivos o falsas asociaciones, que relacionan recompensas con acciones totalmente ajenas a las causantes.

Se implentará a continuación una Deep Q-Network para intentar mitigar estos problemas y no para tanto mejorar los resultados de los 4 primeros escenarios, sino para aprender a jugar a los dos últimos.

AGENT	METRIC	MOVETOBEACON	COLLECTMINERALSHARDS	DEFEATROACHES	DEFEATZERGLINGSANDBANELINGS	COLLECTMINERALSANDGAS	BUILDMARINES
Random Policy	MEAN	1	17	1	23	12	<1
	MAX	6	35	46	118	750	5
Random Search	MEAN	25	32	51	55	2318	8
	MAX	29	57	241	159	3940	46
DeepMind Human Player	MEAN	26	133	41	729	6880	138
	MAX	28	142	81	757	6952	142
Starcraft GrandMaster	MEAN	28	177	215	727	7566	133
	MAX	28	179	363	848	7566	133
Atari-Net	BEST MEAN	25	96	101	81	3356	<1
	MAX	33	131	351	352	3505	20
FullyConv	BEST MEAN	26	103	100	62	3978	3
	MAX	45	134	355	251	4130	42
FullyConv LSTM	BEST MEAN	26	104	98	96	3351	6
	MAX	35	137	373	444	3995	62
Q-Learning	MEAN	26	92	98	83	<1	<1
	MAX	32	107	336	196	<1	<1
SARSA	MEAN	26	93	101	82	<1	<1
	MAX	32	117	336	192	<1	<1

Figura 11.2: Resultados de los agentes incluyendo SARSA.

Capítulo 12

Deep Q-Network

12.1. Sobre redes neuronales

Las redes neuronales[18][19][20] son utilizadas generalmente para tareas de *machine-learning*. Modeladas con inspiración en el funcionamiento del cerebro humano, una red neuronal consiste de una serie de nodos conectados entre si, llamados **neuronas**.

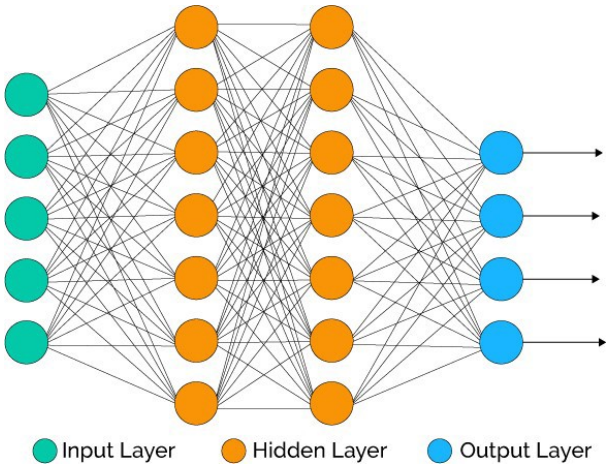

Figura 12.1: Ejemplo de red neuronal.*Imagen extraída de towardsdatascience.com*

Estas neuronas están distribuidas en una serie de capas, la capa de entrada o **input**, la capa de salida o **output** y una serie de capas ocultas o **hidden layers** situadas entre las dos.

Un nodo puede estar conectado a un nodo de la siguiente o anterior capa, a varios o a todos. En general la información va hacia delante, de izquierda a derecha, en un proceso conocido como **feed-forward**. Para el aprendizaje, se hace al revés en un proceso conocido como **back-propagation**, en el que se averigua como afectan los pesos de las conexiones al error, y se intenta ajustar esos pesos. También se conoce como **backward propagation of errors**, ya que se calcula el error de la capa de **output** y se distribuye de vuelta por las capas de la red, con el objetivo de ajustar el peso de las neuronas a partir de calcular el gradiente de la función de error (que calcula la diferencia entre el **output** de la red y el **output** esperado.

A cada una de las conexiones entrantes, un nodo le asigna un peso. Cuando la red está activa, el nodo recibe un dato (número) de cada conexión y le aplica una función que determina si esa neurona se activa (y sigue propagando) o no. Estas funciones se conocen como **funciones de activación**. Algunas de las principales son:

- **Step**: Si el valor está por encima de un límite, la neurona se activa, sino no. Esto produce activación binaria, es decir, la neurona se activa o no se activa pero no puede estar parcialmente activada.
- **Lineal**: Se trata de una función lineal sobre el valor recibido, que proporciona un rango de activaciones. Tiene algunos problemas como que el gradiente es constante y por tanto los cambios realizados por **back-propagation** no dependerán de lo que los cambios en la información recibida por la neurona.

  Además, si todas las capas tuviesen activación lineal, sería lo mismo que tener una única capa con activación lineal.
- **Sigmoide**: Es la función correspondiente a:

  $$A = 1/(1 + e^{-}x)$$

  Esta función no es lineal y sus combinaciones tampoco, lo que elimina el problema de la acumulación de capas con activación lineal. También

proporciona una activación no binaria, y los valores están en el rango (0,1).

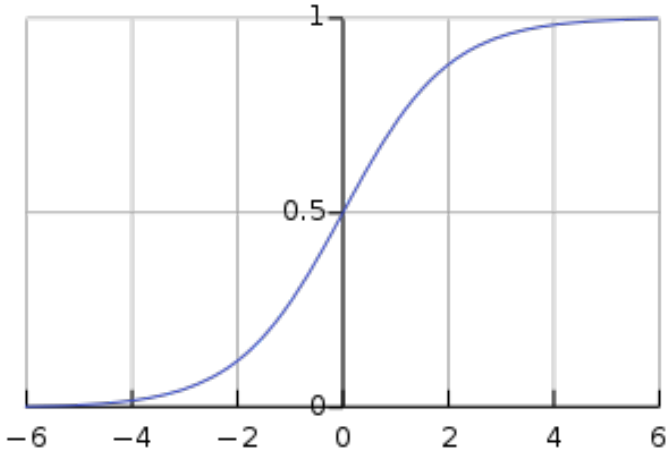

Figura 12.2: Función sigmoide.

El problema principal de la sigmoide es que hacia los extremos, los valores de Y cambian muy poco respecto a los cambios de X, dando lugar a un problema conocido como los "vanishing gradients", caracterizado porque cuando las activaciones alcanzan esa zona, la red no puedo aprender o lo hace extremadamente lento.

- **ReLu**: Se corresponde a:

$$A(x) = max(0, x)$$

ReLu no es lineal y sus combinaciones tampoco. Además es una buena función de aproximación, ya que cualquier función puede ser aproximada mediante combinaciones de ReLu. Además la naturaleza de ReLu hace que las neuronas no se activen para valores negativos de x, dando lugar a una red más ligera. Computacionalmente, también es más barata que la sigmoide debido a que realiza menos operaciones.

El problema principal de ReLu es que en la zona de las X negativas, el gradiente será 0 y eso puede dar lugar a que toda una zona de la red se vuelve pasiva.[21]

12.2. Deep Q-Network

El aprendizaje por refuerzo utiliza normalmente representaciones tabulares para almacenar conocimiento. Representarlo de esta manera consume rápidamente los recursos computacionales, ya que el tamaño de las tablas aumenta exponencialmente respecto al espacio de acciones y estados. Una manera de solventar este problema es utilizar una red neuronal como aproximador, reemplazando a la tabla. Esto facilita la aplicación de los algoritmos a espacios de estados y acciones mayores.

Además, la red facilita la propagación de las estimaciones, ya que cada cambio en un punto específico afecta también a sus vecinos.

En 2015, DeepMind desarrolló un nuevo agente de aprendizaje por refuerzo llamado Deep Q-Network (DQN)[22][23], que entrenaba una ANN (red neuronal artificial) mediante Q-Learning, y era capaz de alcanzar o superar el nivel humano en 29 de 46 juegos de arcade de Atari.

Toda la información principal ya está en el paper de DeepMind, pero para dar una idea general, se resumen los tres puntos principales:

- **Red Neuronal Totalmente Conectadas**: Una red con varias capas totalmente conectadas puede interpretar más información. Sin embargo, también puede sobreajustar dando lugar a resultados nefastos. Habrá que hacer distintas pruebas para encontrar unos parámetros eficaces. DeepMind utiliza capas convolucionales debido a que trabaja con los píxeles de la pantalla. Aquí en cambio se usarán capas totalmente conectadas
- **Experience Replay:** En lugar de entrenar la red directamente con los movimientos que el agente realiza, estas experiencias se guardan en un buffer de tamaño limitado, en tuplas (Estado, Acción, Recompensa, Estado Siguiente). Cada cierto tiempo, se extraen ejemplos aleatorios de este buffer y se entrena con ellos a la red. Algunas ventajas del Experience Replay son un uso más eficiente de la experiencia previa, una de-correlación en el aprendizaje y evitar que la red “olvide”previas experiencias al tener en cuenta las nuevas.

- **Red de objetivo separada:** Los de DeepMind introdujeron el concepto de una red separada que se usaba para generar los Q-values con los que se computa la pérdida para cada acción en el entrenamiento. En teoría, con la segunda red y sus valores fijos, que se actualizan con poca frecuencia, se consigue mayor estabilidad. Inicialmente, se pretendía lanzar entrenamientos sin esta segunda red y luego con ella, para comprobar su eficacia. Sin embargo, por falta de tiempo, esto último se deja como trabajo futuro.

12.3. Los falsos positivos y la asociación de acciones con recompensas

A lo largo de la descripción de escenarios se ha hablado de problemas de falsos positivos y asociación errónea de acciones a recompensas.

Es hora de explicar exactamente cual es este problema. Antes de nada decir que es un problema que se fue detectando según avanzaban los escenarios y se llegaba a los más complejos.

El problema consiste en que el sistema asocia una recompensa con una acción que no tiene nada que ver con ella, y desde luego no es la causante de esa recompensa. Se ilustrará con un ejemplo real que se ha observado en el escenario "Collect Minerals and Gas".

Como breve recordatorio, el objetivo de este escenario era recolectar recursos. Situémonos al principio del escenario, cuando los trabajadores están inactivos. Supongamos que el agente toma la acción de enviar a los trabajadores al mineral. Esta es una buena acción, la mejor de hecho, que le otorgará recompensas al obtener recursos.

Sin embargo, en el momento de hacer la acción, el agente no recibirá ninguna recompensa. Numerosos pasos de juego (muchos) pasarán hasta que en cierto momento, los trabajadores vuelvan al Centro de Mando. Imaginemos que en el paso concreto en el que se obtienen los minerales, el agente se encuentra en un estado **s** y realiza la acción "Seleccionar Centro de Mando". El agente asociará la recompensa con esa acción, y empezará a seleccionar

al Centro de Mando continuamente. Esto confunde mucho a los agentes y dificulta el aprendizaje.

Incluso a pesar de que en las acciones semánticas de más de un paso se tenía en cuenta algo así (no se aprende entre medio de los pasos sino que el estado **s** es el inicial y el final **s'** es después de que el agente haya realizado todos los pasos), esto no es suficiente ya que la recompensa aparece mucho después. Más aún, las recompensas a las distintas acciones tardan distintos tiempos en llegar. Y para dificultarlo más aún, las recompensas a la misma acción desde el mismo estado tardan distintos tiempos en procesarse, afectadas por factores diversos del juego.

Capítulo 13

Agentes de Deep Q-Network

El objetivo de las agentes de Deep Q-Network es intentar aprender a jugar a los dos últimos escenarios. Aún así, se lanzaron también en los primeros cuatro escenarios, pero los resultados son prácticamente idénticos, y si en algun caso el agente de DQN era ligeramente superior (consiguió una puntuación media de 104 en el escenario DefeatRoaches, lo que lo situaría como el mejor agente), fue debido a la aleatoriedad del escenario.

13.1. Collect Minerals and Gas

Se entrenó el agente con el mismo entorno de acciones y estados que los agentes de SARSA y Q-Learning. Inicialmente, la estructura de la red era de una capa de input con tamaño igual a todos los parámetros del estado, dos capas ocultas de 24 neuronas cada una y una capa final de tamaño igual al número de acciones. Esta red sobreajustaba mucho y no producía resultados de ningún tipo.

Se hicieron varios intentos. Primero se redujo el número de neuronas por capa hasta un mínimo de 8 y se fue aumentando de 3 en 3. Además, se experimentó con una única capa oculta en el centro. Se modificaron las funciones de activación y se hicieron experimentos con la función sigmoide

para las capas ocultas, e incluso un experimento con todas las capas con activación lineal.

Tras varios intentos, la red final tenía las mismas capas de input y output, pero entre medio dos capas ocultas de 14 nodos cada una con activación ReLu y activación lineal para la última capa.

Este agente consiguió resultados por primera vez, con una media de entre 2100 y 2600 puntos. Esta puntuación seguía siendo inferior a la de un jugador humano y a la de los agentes de Deepmind.

La estrategia que aprendió el agente consistía en enviar a los trabajadores a una línea de mineral y no hacer nada más. Las fluctuaciones venían de que a veces los enviaba a la línea de mineral de la derecha, que está más alejada del Centro de Mando, lo que hacía menos eficiente la recolección.

El agente no era capaz de aprender a asociar acciones como construir más trabajadores con la recompensa. Sin poder solventar este problema, se optó por otro enfoque.

Ya que el agente no es capaz de asociar acciones con sus recompensas correctamente, ¿por qué no intentar que aprenda que estados son mejores?

Se añadió un parámetro extra al estado, indicando el número de trabajadores existentes. Este es un parámetro continuo, lo que incrementa mucho el número de estados. La idea era que el agente aprendiese que en estados con más trabajadores, obtenía más recompensa (ya que había más de ellos recolectando).

El agente aprendió a construir trabajadores a ritmo constante, y además aprendió que el estado **supply_block** no le permitía llegar a los estados con más trabajadores, así que lo evitaba construyendo depósitos de suministros.

Si bien es cierto que el agente hacía muchas acciones sin sentido y que había muchos trabajadores construyendo depósitos o barracones de más, el hecho de que en general hubiese muchos trabajadores hacía que recolectase muchos más minerales, obteniendo una media de unos 4100 - 4200 puntos, con un máximo de 5200, colocándose bastante por encima de los de DeepMind, aunque aún bastante por debajo de un jugador humano.

13.2. Build Marines

A pesar de que DQN consiguió aprender el escenario anterior, no tuvo éxito con este. El agente nunca consiguió averiguar el camino ni los requisitos que lo llevaban a construir marines, y tampoco logró asociar la acción **_BUILD_MARINE** con la recompensa.

Se hicieron distintas pruebas, tanto modificando los parámetros de la red, e incluso un enfoque parecido al que se usó para resolver el escenario anterior, añadiendo a la definición de estado un campo con el número de barracones, para ver si relacionaba los estados con más barracones con una mayor recompensa.

Sin embargo, todos los intentos fueron fútiles y el agente no logró aprender.

Capítulo 14

Conclusiones

Tras todo el trabajo realizado, podemos concluir que los agentes han tenido un éxito aceptable. Han obtenido resultados similares a los de DeepMind, e incluso mejores en el quinto escenario, que era de los más complejos. Algo decepcionante es el fracaso en el último escenario, aunque era de esperar.

Los algoritmos probados en este proyecto no son lo suficiente potentes para resolver el último de los escenarios, pero en general, no parece haber una respuesta clara al problema de la asociación de recompensas y falsos positivos. Utilizar algo como un SARSA de n pasos tampoco sería una solución debido a que cada acción dura un número indeterminado y variable de pasos.

Una alternativa podría ser tener distintos agentes cada uno centrado en una tarea aún más específica y luego unirlos. El objetivo sería que cada gente se centrase en una recompensa de un tipo específico. Aún así, mi opinión es que el problemas de los falsos positivos sería presente.

Otra opción planteada, extremadamente costosa en tiempo, consistiría en aislar las acciones del agente. Realizar una única acción, esperar un tiempo elevado y almacenar la recompensa. Después probar con otra, y almacenar la recompensa, y así. Ir repitiendo este proceso con distintas acciones y estados y utilizar alguna función para ir decrementando el tiempo, de forma que al final el tiempo entre acciones fuese pequeño y se ejecutasen acciones seguidas que, por su cuenta, habían resultado efectivas.

Sin embargo, estas ideas no son más que especulaciones y sería necesaria mucha más investigación para llegar a algo concreto.

Concluir diciendo que para tareas más sencillas el aprendizaje por refuerzo es una gran alternativa (como puede comprobarse con el éxito de escenarios como *Move to beacon* o *Defeat Roaches*), y especialmente cuando las recompensas se reciben tan pronto como sea posible. Aún queda sin embargo un largo camino hasta que el aprendizaje por refuerzo consiga dominar problemas del calibre de *Starcraft 2*.

14.1. Opinión personal

Trabajar en este proyecto ha sido una gran experiencia personal. *Starcraft 2* es un videojuego que me dio grandes momentos de diversión en mi infancia/adolescencia, y poder terminar los cuatro años de grado con un proyecto de IA sobre el mismo ha sido muy satisfactorio.

Además, aunque ya estaba interesado en la inteligencia artificial, este proyecto me ha permitido descubrir un nuevo campo de la misma con el que no había trabajado en el grado, y que me ha encantado. He aprendido muchísimo sobre aprendizaje por refuerzo, desde sus características hasta 3 nuevos algoritmos y variantes de los mismo.

Más aún, ha sido un gran ejercicio de razonamiento. En cierta manera, lo describiría como detectivesco, ya que ha involucrado mucha prueba y error, intentando averiguar que confundía a los agentes y que modificaciones podía hacer para mejorar su juego.

En definitiva, estoy muy contento con el resultado, que si bien no ha logrado puntuaciones estratosféricas (y tampoco era el objetivo), ha conseguido igualar o superar los *benchmarks* establecidos, definir las principales limitaciones y causas de problemas, y ha resultado en un gran proceso de aprendizaje.

Capítulo 15

Trabajo futuro

Como trabajo futuro para este proyecto se proponen diversas opciones:

- **Experimentar con mejoras de DQN y A3C:** Además de añadir la segunda red a DQN, existen mejoras para como por ejemplo Double DQN, que trata de ajustar la sobreestimación que suele hacer DQN utilizando la segunda red para obtener los Q-values correspondientes a la acción por la primera red.

  Otra variante es Dueling DQN, que descompone el valor Q(s,a) en V(s) (como de bueno es estar en un estado) y A(a) (como de buena es una acción respecto a las demás), combinando estos dos valores en la última capa, con el objetivo de que la red obtenga mejores estimaciones de los valores de un estado sin necesidad de asociarlo a una acción. Esto podría ser útil sobretodo en los últimos dos escenarios.

  Por último, experimentar con el algoritmo de vanguardia de DeepMind, A3C, para ver hasta donde puede llegar.

- **Investigar sobre nuevos algoritmos:** A pesar de las mejoras mencionadas, nuevos algoritmos serán necesarios para avanzar en problemas tan complejos. Se propone como posible trabajo futuro a medio o largo plazo adquirir nuevos conocimientos y profundizar en la investigación de este campo.

Bibliografía

[1] Georgios N. Yannakakis and Julius Togelius. *Artificial Intelligence and Games*. Springer, Cham, 2018.

[2] *Wikipedia - Starcraft 2*. https://es.wikipedia.org/wiki/StarCraft_II:_Wings_of_Liberty.

[3] *Web oficial de Starcraft 2*. https://starcraft2.com/es-es/game.

[4] *Teamliquid.net*. http://www.teamliquid.net/.

[5] *Wikipedia - Aprendizaje por refuerzo*. https://en.wikipedia.org/wiki/Reinforcement_learning.

[6] *Github - Reinforcement Learning*. https://github.com/aikorea/awesome-rl.

[7] *Deepmind*. https://deepmind.com.

[8] *Blog de Deepmind sobre Deep Reinforcement Learning*. https://deepmind.com/blog/deep-reinforcement-learning/.

[9] *Artículo sobre usos del aprendizaje por refuerzo*. https://www.technologyreview.com/s/603501/10-breakthrough-technologies-2017-reinforcement-learning/.

[10] Michal Certicky and David Churchill. *The Current State of Starcraft AI Competitions and Bots*. 2017.

[11] *Curso de Deepmind sobre Reinforcement Learning*. https://www.youtube.com/watch?v=2pWv7GOvuf0.

[12] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, 2017.

[13] *Artículo sobre Q-Learning.* ://medium.com/@curiousily/solving-an-mdp-with-q-learning-from-scratch-deep-reinforcement-learning-for-hackers-part-1-45d1d360c120.

[14] *Tutorial sobre Q-Learning.* http://mnemstudio.org/path-finding-q-learning-tutorial.htm.

[15] *Introducción a varios algoritmos de aprendizaje por refuerzo.* https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-sarsa-dqn-ddpg-72a5e0cb6287.

[16] *Artículo de wikipedia sobre SARSA.* https://en.wikipedia.org/wiki/State

[17] *SARSA vs Q-Learning*. https://studywolf.wordpress.com/2013/07/01/reinforcement-learning-sarsa-vs-q-learning/.

[18] *Artículo sobre redes neuronales*. https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e.

[19] *Artículo sobre redes neuronales*. https://www.explainthatstuff.com/introduction-to-neural-networks.html.

[20] *Artículo sobre redes neuronales.* http://news.mit.edu/2017/explained-neural-networks-deep-learning-0414.

[21] *Artículo sobre funciones de activación en redes neuronales.* https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0.

[22] *Artículo sobre DQN*. https://towardsdatascience.com/welcome-to-deep-reinforcement-learning-part-1-dqn-c3cab4d41b6b.

[23] *Post de DeepMind sobre DQN*. https://deepmind.com/research/dqn/.